

FOG – Applying Blockchain to Secure a Distributed Set of Clusters

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS, COMPUTER NETWORKS AND
DISTRIBUTED SYSTEMS, FACULTAT D'INFORMÀTICA DE BARCELONA (FIB), UNIVERSITAT
POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Pau Marcer Albareda

Date: 01-05/07/2019

Tesis tutor: Beatriz Otero Calviño

Tesis supervisor: Eva Marin Tordera

Arquitectura de Computadors (AC)

INDEX

Abstract	4
Keywords	4
1 Introduction, motivation and objectives	5
1.1 Introduction.....	5
1.2 Motivation	5
1.3 Objectives	6
2 Background	7
2.1 Fog Computing	7
2.1.2 Fog scenario in this project.....	8
2.2 Blockchain	8
2.2.1 Functionality Overview	8
2.2.2 Blockchain Architecture.....	9
2.2.3 Block structure.....	13
2.3 Consensus.....	14
2.4 Security models overview	17
2.4.1 Introduction.....	17
2.4.2 PKI (Public Key Infrastructure) [15]	17
2.4.3 Kerberos [16]	17
2.4.4 BCTrust: A decentralized authentication blockchain-based mechanism [17]:.....	18
2.5 Blockchain Frameworks.....	19
3 Security challenges on the Fog.....	20
3.1 Mobility issues.....	20
3.2 Centralized security models	20
3.3 Centralized data.....	20
4 Applying Blockchain to Secure a Distributed Set of Clusters	21
4.1 Objective	21
4.2 Components.....	26
4.2.1 Secure Edge Node (SEN).....	26
4.2.2 Device	26
4.2.3 Gateway	26
4.2.4 Simple Device	26
4.2.5 Anonymous device	27
4.2.6 Resume of component characteristics	27
4.3 Proposed architecture.....	29

4.3.1 Network.....	29
4.3.2 Blockchain: Blocks.....	30
4.3.3 Blockchain: Consensus	33
4.3.4 Webinterface.....	36
4.3.5 Architecture integration	39
4.4 Functionality	41
4.4.1 First key distribution.....	41
4.4.2 SEN0	41
4.4.3 Sens.....	42
4.4.4 Devices	46
4.4.5 Authentication.....	50
4.4.6 Validation	51
4.4.7 Extensions.....	52
4.6 Programming language used.....	53
4.7 Prototype code structure	53
5 Validation of the use cases	54
5.1 Methodology.....	54
5.2 Validation.....	55
5.3 Testing scenario replication	63
6 Future work	63
7 Conclusions	64
8 Acknowledgments.....	65
9 References	66
10 Annex.....	67

ABSTRACT

5G has already been presented and shown in major congresses, thus it's just a matter of time that this technology comes to the enterprise and public usages. Therefore, we can affirm that we are moving towards a world where everything will be connected (i.e. our cars, our houses, our wearable devices). Consequently, the number of devices connected to the Fog will be around the billions. This drastic increase in connected devices at the Fog layer promotes a change in the Internet architecture, that requires new technologies to manage the newly Fog devices. Despite the cloud being a powerful model, at the end it's a centralized architecture, thus it fails to scale with the addition of millions of Fog devices. The Fog presents a great amount of architectural and management challenges, such as who will run the Fog infrastructure, or how the Fog nodes will provide services by themselves, and a sometimes forgotten but critical aspect, the security. The current centralized security architectures do not scale well enough in order to be applied on the Fog. Those models such as Certificate Authorities (CA's) are centralized, usually on cloud providers, and offer a much more static security (i.e. a website secured with a CA, that barely changes IP and doesn't move). At the end this kind of security approaches are invalid in environments where devices are moving and changing networks constantly, like the fog scenario, because the same nature of the approach makes it invalid for such scenarios.

Therefore, we require new and completely distributed security architectures, capable of being flexible and scalable, while at the same time providing fault proof security to the Fog. A new technology that has been growing lately is the blockchain, this technology really shines on completely distributed systems. The blockchain is capable of keeping an immutable set of data distributed across multiple peers on a network. Then the peers can use that data and update it through a consensus procedure, performed following a consensus algorithm criteria.

The main objective of this project is the proposal of a novel blockchain architecture that will contain all the Fog session information. This information will be used to provide security to the Fog devices, and it will be capable of providing authentication and verification mechanisms to those devices, ensuring the integrity of the data provided. To accomplish this objective a Fog profile will be used. Each new device will be able to register its profile in the Fog session, with a required set of public keys, and by providing an extensible Rule-set field that will contain all the required information in order to identify a device (i.e. the device specs such as CPU or RAM), to later on execute services on the Fog, based on such specs and information.

Keywords

Blockchain, Distributed ledger, Fog, Edge, Security, Authentication, Session architecture, Device profiling.

1 INTRODUCTION, MOTIVATION AND OBJECTIVES

In this first section the project will be introduced with its motivation and an overview of the projects objectives.

1.1 Introduction

With the arrival of 5G, the increase on smart car production and the latest smart house devices that can be actually bought from most relevant stores, we can affirm that we are moving towards a world where devices will be everywhere and everything around us and will have sensors. Consequently, there will be a major number of devices connected to the Internet. This huge amount of devices can be more than just connected devices. They can be part of a Fog network able to self-orchestrate and provide services at the Fog level, without relying on the centralized cloud services. But this proposes a change in the actual Internet architecture used to provide services that has been used since the first days of the net, from centralized to distributed. This change towards a decentralized Internet presents a whole set of new challenges, such as who will be the owner/s of the Fog infrastructure or who will be the Fog operator/s. Moreover, it presents the need for a new decentralized security architecture, as the actual security architectures are based on centralized cloud services, thus are unreliable and do not scale well enough for a Fog system. In addition, we believe that for the Fog to become a valid and used technology, it has to be able to orchestrate itself, security included, without completely relying on the cloud, thus introducing the need of a new completely distributed security architecture, because if the Fog relies on the cloud in order to securely operate, it's not Fog at all, it's just an extension of the cloud.

1.2 Motivation

Nowadays there is no distributed architecture that can be used to secure a Fog area. We define a Fog area as any area with enough IoT devices to form a cluster (i.e. Building with sensors, cameras and smartphones). All of the available technologies highly rely on centralized cloud services, such as Certificate Authorities (CAs) and its certificates or centralized profile servers. The motivation of this project is to fill this technological gap, for this reason we propose a novel distributed security architecture, that could be applied on Fog areas in order to make them secure. To do so, we will use the newly and promising blockchain technology.

The blockchain technology provides a distributed and immutable set of data in form of transactions, distributed across all peers in a blockchain network. Usually blockchain is associated with cryptocurrencies, but blockchain has much more uses than that, for example Sovrin[1] a distributed identity platform, or the projects under the Hyperledger framework[2]. Blockchain is a relatively new technology thus its usages are still being explored. We aim to test this technology on a simulated Fog scenario, and validate if it's a valid technology in order to secure it.

1.3 Objectives

The final objective of this project is to propose a new security model for the Fog using blockchain technologies. We will define and deliver the architecture for the security model plus the prototype, which will prove if the proposed model is valid in order to provide security to the Fog. The objective of the architecture is to define all the required aspects of the security model to be applied. On the other hand, the objective of the prototype is to prove the functionality of the proposed security model. The resulting project will be able to verify each device identity in the Fog system, establish secure connections between authenticated devices and verify the integrity of the data provided by the Fog devices, while being completely distributed and cloud/connection independent.

In order to accomplish the previous objectives, we propose to use the blockchain as a session holder for the Fog. Therefore the blockchain will store all the relevant information for the Fog session, to provide authentication and security to the Fog devices. Since all the actions performed on the blockchain are called transactions, we will call a transaction to each action performed on the Fog session. The transaction will have a public key, which will identify the device related to the transaction, and a set of actions performed on the session, we call this set of actions the rule-set (i.e. registration of a device). At the end, this project will be the beginning of a base where to build a secure Fog platform. The project will provide the base authentication and tools in order to secure a huge amount of Fog projects. Moreover, we expect the rule-set to be extendible with custom fields, so this project can fit in almost any Fog related project out there, which needs a distributed security architecture.

Overview of objectives:

- Proposing a new blockchain model to secure a Fog area.
- Delivering the architecture plus the prototype code for the proposed model.
- Using the prototype to test the model validity with a series of testing and validation scenarios.
- The model must secure a Fog area, and provide Fog profiles to the Fog devices.
- The blockchain will be the Fog session holder.
- All the devices must be able to register into the session.
- All the devices must be able to use the session to establish secure connections.

2 BACKGROUND

In this chapter, we will present the background for this project. The content goes as following: In section 2.1 we will present a brief overview of the Fog paradigm and we will introduce the most related concepts of Fog with our project. Next, on section 2.2, we will present the blockchain technology, with a review of functionality, architecture and the standard block structure. Then, in section 2.3, we will review the most relevant consensus algorithms for this project, and we will point the best candidates for later using in our architecture. The 2.4 point reviews the security models that can be used nowadays on any architecture, and a proposal that uses the blockchain to secure devices, but that follows a different approach. Finally on the point 2.5 we will review the main blockchain frameworks that could be used in order to implement our architecture into a prototype.

2.1 Fog Computing

With the arrival of 5G, it's just a matter of time that Fog computing becomes a hot topic, if it's not already one. We cannot deny that we will need new architectures in order to manage all the upcoming IoT devices, and that those architectures will have to be based on distributed systems, as it's not feasible to keep the same centralized cloud architecture to manage billions and billions of devices.

Fog computing[3] was first mentioned by CISCO and its base proposal is an infrastructure at the edge of the network, capable of using the resources at the extreme of the network. This infrastructure is composed by smart grids, smart cars and a huge diversity of connected devices.

Since the Fog infrastructure is placed near to the user, it will be capable of providing a near zero latency connection, that for example could be used to perform real time video streaming, or augmented reality support to its users.

This opens the Fog for a new branch of operators, which could compete in order to provide services on the Fog infrastructure. Fog is tied to the cloud, but in our opinion, in order to survive it should not rely on the cloud for everything, as this makes the cloud a better choice for most users and services. The Fog should be able to orchestrate itself and the services it provides to the users, and then and only when a service needs it should be scaled to the cloud. But this scaling decision should be taken on the Fog and not on the cloud.

2.1.2 Fog scenario in this project

Our project is focused on providing a distributed authentication model to any distributed architecture, but in the Fog is where this kind of approach makes more sense, as the amount of devices will be high, heterogeneous and constantly growing and moving, a centralized architecture is not suitable for this kind of environment.

The Fog is meant to execute and provide services closer to the user, compared to the cloud, and those services will need a valid trust model, in order to correctly operate without security vulnerabilities that will compromise the whole system. The model used to provide security to the Fog can be this project proposal, as this project target is enabling security on distributed systems and it's intended to operate on any Fog environment, thanks to its extensible properties. Therefore, the project can be extended in order to provide security to the services executing on the Fog, even if the Fog is composed by various different subsystems. In conclusion we present the Fog as the main objective architecture for this project to secure, but this doesn't mean that this project can only be used on Fog scenarios, instead this project can be used on other kind of distributed architectures, where security is required, and devices can have a profile.

2.2 Blockchain

Blockchain technology is a trend nowadays, as more researchers are exploring its capabilities applied to a huge amount of different use cases. Indeed, it's really powerful when applied to highly distributed systems, as the blockchain is completely distributed while at the same time its data is immutable.

2.2.1 Functionality Overview

Blockchain is a technology found in distributed systems, it enables multiple peers to keep a copy of the same data-set across the network. The data-set is formed by blocks, and these are linked to each other by the previous block hash, thus forming a chain that cannot be altered. Usually blocks are signed by one or multiple peers depending on the consensus protocol used (see illustration 1).

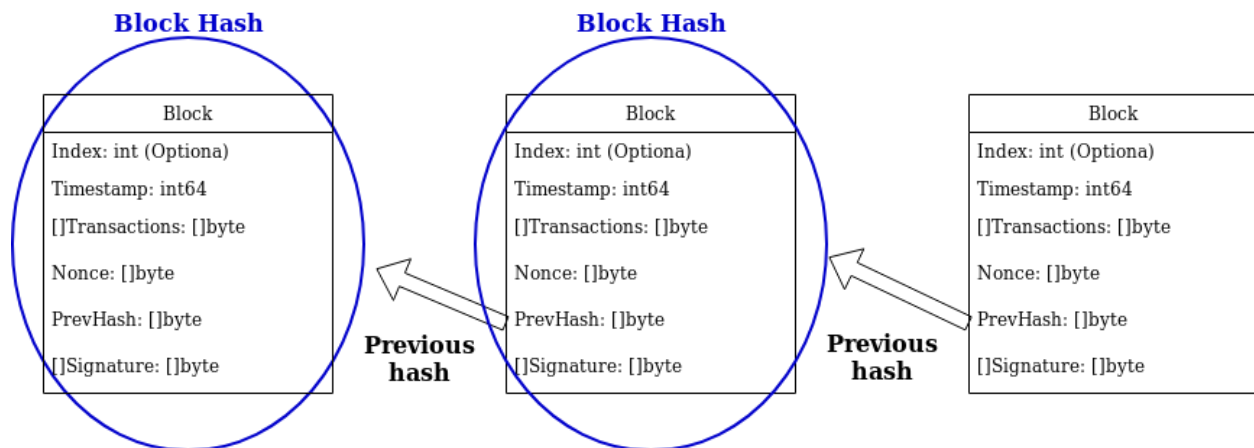


Illustration 1: Blockchain linked blocks by previous block hash

Once we have the linked blocks, we can start storing data in those blocks, that data is usually referred as a transaction, since the blocks have a timestamp, and no block can be deleted, it creates an immutable register of transactions. This is especially useful in cryptocurrencies, as it is used to store financial operations. However, it can be used to store any kind of data. In a high level we can divide the blockchain in two parts:

The block structure, explained above, that ensures the integrity of the blockchain data and defines which data we can be stored as a transaction.

The consensus protocol, which distributes the blockchain over the peers and enables the consensual addition of new blocks or transactions, into the blockchain, ensuring the distribution and integrity of the blocks across all peers.

2.2.2 Blockchain Architecture

Blockchain architecture can be cut into three main architectures: Permissionless, permissioned, and hybrid or consortium. Diverse blockchain frameworks use different architectures depending on their final objective. For example, Ethereum[4] and Bitcoin[5] use a permissionless architecture, while on the other hand, Hyperledger Fabric [2] supports consortium architectures. There aren't any major blockchain framework that use a permissioned architecture[6], as some argue that it goes against the blockchain basics.

On the following points we will introduce and overview the three main blockchain architectures. In point 2.2.2.1 we introduce and review the permissionless blockchain architecture. Later on in point 2.2.2.2 we introduce and review the permissioned blockchain architecture. Finally in point 2.2.2.3 we introduce and review the permissionless blockchain architecture.

2.2.2.1 Permissionless

In the permissionless blockchain architecture, anyone can join the blockchain network and participate in the consensus, since there is no authentication required against other devices when performing consensus operations. Illustration 2, illustrates the permissionless architecture, being the blue cloud the blockchain network, and the blue nodes the devices belonging to it. As defined in the permissionless architecture all nodes can participate in the consensus.

This architecture perfectly suits anonymous cryptocurrency networks where participants want to be anonymous to each other, and is the basis of the first blockchain ever, that is bitcoin. Well-known consensus algorithms belonging to this architectural group used on this kind of blockchain networks are Proof of work and Proof of Stake[7].

Blockchain network

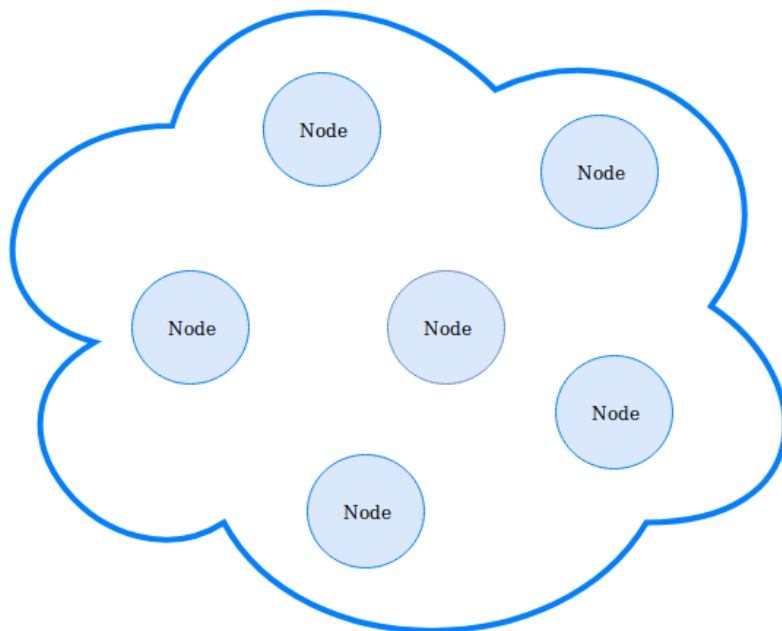


Illustration 2: Public blockchain

2.2.2.2 *Permissioned*

Unlike the permissionless architecture presented in the subsection above, in the permissioned architecture, the access to the blockchain network is controlled and, consequently, only authenticated devices can join the blockchain network. Illustration 3, depicts the permissioned blockchain network, and shows the two main important differences vs illustration 2. First, while similarly to the permissionless architecture shown in illustration 2, the blue cloud, represents the network, and the blue nodes represent the devices belonging to the blockchain network, additional red nodes come up representing the set of nodes outside the private network not enabled to access the blockchain. Second, in this permissioned architecture approach only the blue validator nodes can participate in the consensus.

In this kind of architectures, consensus is controlled by one single organization, putting together either one node or a set of nodes to manage all consensus related tasks, depending on the used consensus algorithm. This makes this architecture a non-optimal choice for scenarios where a large set of distributed nodes and multiple networks are together. In addition the fact that the consensus it's controlled by a single organization means that such organization will have all the power on the blockchain and this is a bad approach on blockchains.

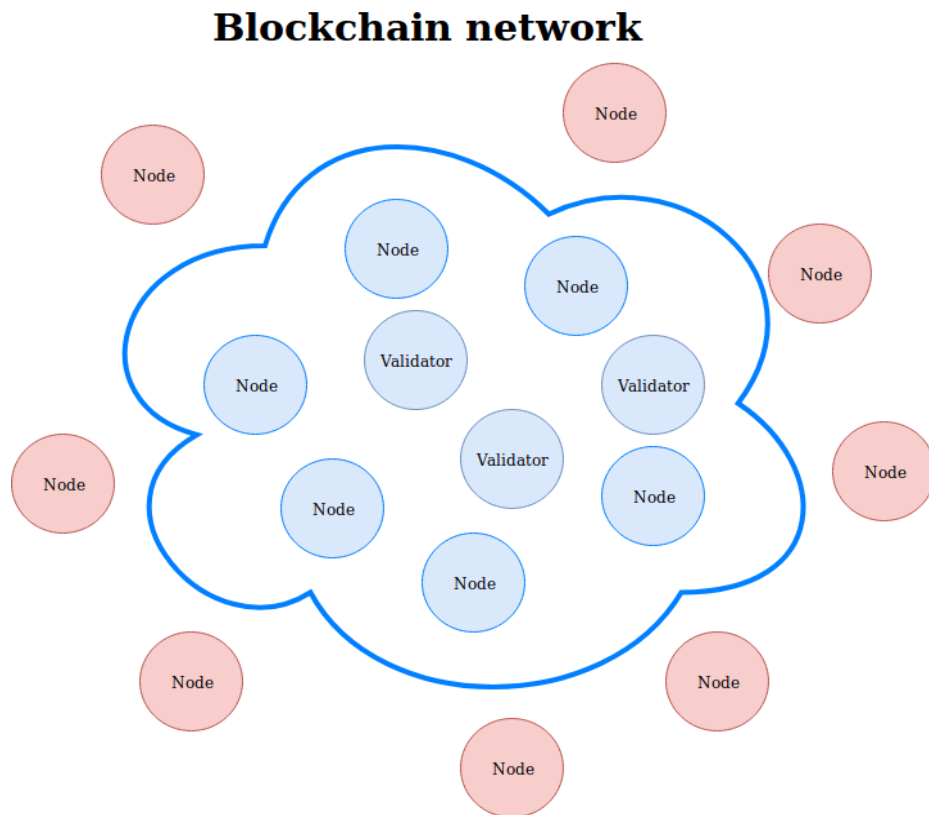


Illustration 3: Permissioned blockchain

2.2.2.3 hybrid/Consortium

In this architectural approach, any device is allowed to read the blockchain, any device can verify its data, but only a controlled set of 'validators' is allowed to perform consensus operations and commit blocks. Similarly to the two previous architectures, illustration 4 represents the blockchain network as the blue cloud and the devices belonging to it as blue nodes. As in illustration 2 (i.e. permissionless architecture), any device can join the network, but differently, not all devices can participate in the consensus. Indeed, two types of devices may be defined, normal devices (belong to the blockchain network but that can't participate in the consensus), and validators (that can participate in the consensus)[7].

Acting as a hybrid approach between the two described before, seems a reasonable solution to provide security with no need to lose the complete control. Indeed, this approach enables all network devices to review and verify the blockchain data, while simultaneously enabling a consortium of organizations to control the blockchain through a set of validators owned by the consortium members. By doing so, the blockchain cannot be altered by any attacker as long as the majority validators are secure (this majority is specified by the consensus algorithm).

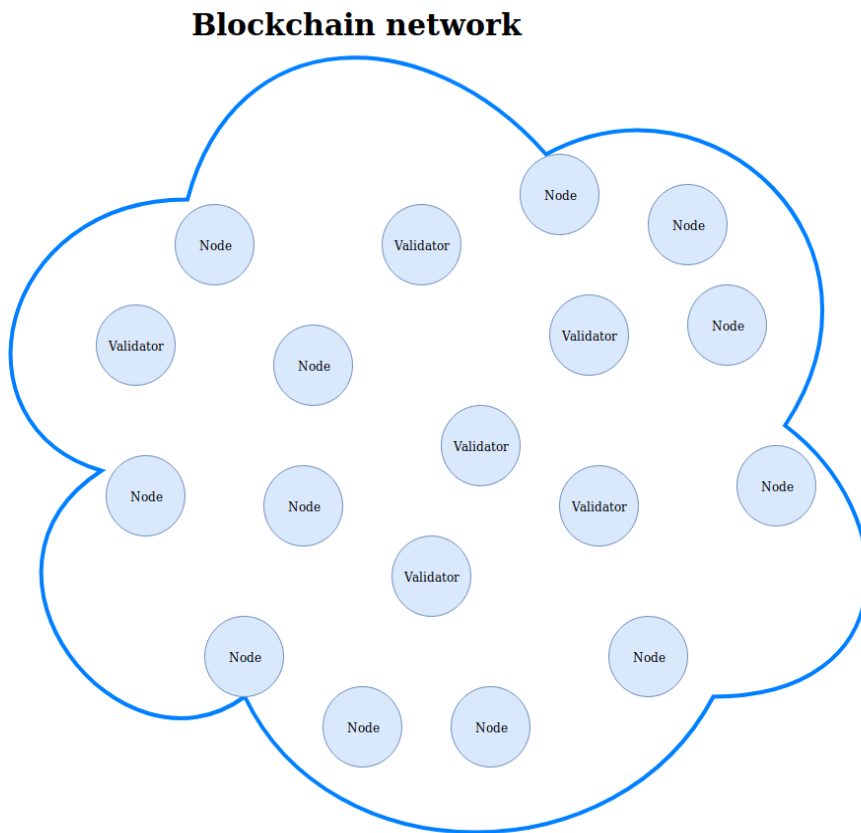


Illustration 4: Consortium blockchain

2.2.3 Block structure

The block structure is the data that a block contains, and it completely defines the characteristics of the resulting blockchain, the transactions it can accept and the security of those transactions.

Depending on the final purpose of the blockchain, the block structure will have to be modified in order to fulfill such purpose. The definition of the schema is something that cannot be altered in the future, it is preserved with the blockchain. The block structure is highly related to the consensus algorithm used, as it has to tolerate such protocol.

There are some common block fields that are present on almost all the blockchains (see illustration 5), those fields are defined as following:

Block
Index: int (Optional)
Timestamp: int64
[]Transactions: []byte
Nonce: []byte
PrevHash: []byte
[]Signature: []byte

Illustration 5: Blockchain block example

Index, although this one is not strictly required, most blocks contain its index inside the blockchain. The index is used to help store the block following an order and help fasten query operations.

Timestamp, it contains the time of creation of a block. This is done to force an order on the blockchain, and keep control of the timings, usually to know when a transaction had happened, and to establish a time order between blocks and block transactions.

Transaction, The data payload of the block. The information contained here might change depending on the blockchain purpose. The transaction must contain at least one signature from the issuer of the transaction, and a payload of data that could be almost everything.

Nonce, blocks may contain this field in order to increase security. This field is a random array of bytes that introduces noise to the block in order to strength its cryptographic hash, it's most useful in small or repetitive blocks where the hash function can be weaker.

Previous block hash, which stores the hash of the previous block. This is done to keep the blockchain integrity safe, as there is no previous block that could be altered, even with the signing keys, without creating an incoherence in the following block that could be easily detected by any participant device.

To obtain the block hash, a hashing algorithm is used. The block hash is obtained from the block headers, which usually take all the fields inside the block.

Signature, the block must contain at least one signature from the validator that creates it in the consensus phase, to verify the block and know who issued it.

2.3 Consensus

A consensus algorithm is a mechanism used to keep a set of data distributed across multiple network peers, and controls the data addition and distribution.

Blockchain is the perfect example of a distributed system where various consensus algorithms can be applied, resulting in the same result or state for the system, a consensus between the nodes on a certain action, usually which transaction or set of transactions should be applied to the next block. But to achieve that result, there is a huge amount of algorithms following different strategies that, as everything in architecture design, have a tradeoff. Some of them are faster but not great fault tolerant, while others have a logarithmic overhead of communication messages but are byzantine fault tolerant. Moreover, some of them are permissioned while others are open to everyone.

In resume, one of the things that distinguish each blockchain framework is the use of different consensus algorithms. For example, Hyperledger Indy[8] uses a modified implementation of the RBFT consensus algorithm[9], Tendermint uses the Tendermint consensus algorithm[7], Hyperledger Fabric uses the PBFT consensus algorithm[10] while other blockchain frameworks more related to cryptocurrencies use more generalized algorithms, such as Ethereum[4] that uses Proof of Work[11] on its mainline and its testing the usage of Proof of Stake[11] and Bitcoin[5] that uses Proof of Work.

The objective of the following sections is to introduce the most know or common consensus algorithms, in order to later on choose one consensus algorithm or algorithms for our proposal.

We can abstract the consensus algorithms and divide them into two subgroups: the ones that are open, and allow everyone to participate into the consensus phase, and the ones that are permissioned and require the peers to meet a certain requirement in order to join the consensus phase [7].

Property	PBFT	RBFT	Zyzyva	Tendermint	PoW	PoS
Peer Identity management	Permissioned	Permissioned	Permissioned	Permissioned	open	open
Fault Tolerance	< 33.33 % byzantine voting power ²	< 33.33 % byzantine voting power ²	< 33.33 % byzantine voting power ²	< 33.33 % byzantine voting power ²	< 25% computing power	< 51% stake
Optimal power usage	yes	yes	yes	yes	no	partial
Example	Hyperledger Fabric [2]	--	--	Tendermint [12]	Bitcoin[5]	Peercoin[13]

Table 1: Comparasion of the consensus algorithms properties.

²Byzantine voting power: The minimum required nodes to fail in order to alter a byzantine fault tolerant algorithm.

PBFT Practical Byzantine Fault Tolerant [10]

PBFT is the most basic Byzantine fault tolerant protocol, the following protocols Zyzzyva and PBFT part from this protocol base.

This byzantine fault tolerant protocol is distributed in rounds, each adding a new block. The process starts by first, selecting a primary out of the set of possible validators. To that end, some policies are applied customized to the final objective and environment of the protocol. Once the primary is selected, it becomes responsible for setting the order for the different transactions as they are being received by the primary. Then, when the primary receives a petition to add a new block, the primary runs the commit protocol, which is structured into three different phases, pre-prepared, prepared and committed, moving a necessary condition for a node to pass each phase to receive votes from 2/3 of the validator nodes. Certainly, the PBFT protocol requires every node to be known by the blockchain network.

This protocol could be usable by our project, as it fits into our needs of a permissioned Byzantine fault tolerant protocol, but in our opinion there are similar options that offer better performance as we will see in the following points.

Zyzzyva: Speculative Byzantine Fault Tolerance [14]

Zyzzyva protocol introduces the usage of speculation in order to reduce the cost and simplify the design of byzantine fault tolerant algorithms. The clients respond automatically to any requests, skipping the three phase commit proposed in PBFT. To do so, they follow the order of the so called 'primary', thus minimizing the network communications to the theoretical minimums. However some inconsistencies may come up on the system, those inconsistencies must then be addressed by the clients.

This protocol could be usable by our project, as it fits into our needs of a permissioned Byzantine fault tolerant protocol.

RBFT Redundant Bizantine Fault Tolerant [9]

RBFT protocol targets the performance problem when the faulty or malicious replica is in fact the so called 'primary'. To do so, it makes all the clients to be a 'primary', and introduces the concept of the 'master instance'. All validators/primary instances are responsible to order the requests, but only the master instance order is actually executed. The results presented in [9] are promising as the performance when no failure occur is similar to the top protocols, and when a failure occurs, the loss of performance is only a 3%, while in other protocols is a major 78%.

This protocol could be usable by this project, as it fits into our needs of a permissioned Byzantine fault tolerant protocol and it's actually far better in performance than zyzzyva if a failure occurs.

Tendermint [12]

This protocol is based on the byzantine fault tolerant consensus protocol idea, it uses a set of validators that sign blocks, and blocks require a set of signatures in order to be committed. The consensus phase consists of rounds, each round has three steps: Propose. Prevote and Precommit. Additionally, validators issue three different votes: Prevote, Precommit, Commit. A block is committed by the network when in a round there are 2/3 of majority of commits.

This protocol is permissioned because in order to become a validator, a user first must put coins in a bound deposit. But, anyone can be a validator if the coins are deposited. This is good for public blockchains, but is not that suitable by our blockchain goals, as this project demands a total control on validators.

PoW Proof of Work [11]

In proof of work consensus algorithms, there is the concept known as miners. Miners are the peers that try to create or mine new blocks in order to be added to the blockchain. In order for those blocks to be accepted by others they require to meet a special hash, starting with 4 consecutive zeros, thus miners must try different combinations of transactions and calculate multiple hashes of a block since they find the required pattern. Once the pattern is found, the block is distributed by others and committed. This peculiar consensus algorithm requires a high amount of computational power, as by 2019 this consensus algorithm is using more power than some small countries.

This protocol requires too much computational power and is only usable on totally public networks. In conclusion, it's not valid for our needs.

PoS Proof of Stake [11]

This consensus algorithm rewards the users with most stake in the system, (i.e. usually more money), and allow those users to validate or create new blocks from the transaction set. This approach follows the theory that the users owning the most part of the money are the ones that one to keep the network alive and won't go against it, as they are the ones with more to lose.

As PoW algorithm, this protocol is only usable on totally public networks and it's not valid for our needs.

2.4 Security models overview

2.4.1 Introduction

In this chapter, we aim to present some of the most used and relevant (to our project) security models on the cloud and its limitations within a Fog environment. The first model to be presented will be the Public Key Infrastructure or PKI. This is the actual model that provides security to most of the Internet websites right now. Another model we will review is the Kerberos authentication server, which is an example of a centralized authentication cloud server. At the end, we will review a model that makes use of blockchain in order to provide decentralized authentication.

2.4.2 PKI (Public Key Infrastructure) [15]

One of the most used security models nowadays on the cloud is the PKI (Public Key Infrastructure). PKI model is usually based on certificates, where each certificate has a public key associated to a domain name or IP and the whole certificate is signed by one or multiple CA (Certificate Authority). Thus if a client trusts a CA, then it trusts that when a domain provides a public key with a certificate and that certificate is signed by a trusted CA then the domain is trusted. Therefore, certificates are really good in providing security on the cloud. However, certificates are linked to domain names to verify the identity of the machines and can't provide any extra information on a host. Hence CAs are not suitable for the Fog, due to the highly mobility of the devices and should not be linked to domain names.

Even though some can argue that certificates could be customized in order to keep the required information for a Fog device, we would still have the problem of a centralized CA server being a bottleneck, and the key distribution to the Fog devices in order to trust those CAs won't be as easy as done in the cloud. The same strategy can't be followed in the Fog since the keys are embedded in the browsers and verified by the browser providers. In resumes PKI is too static to fit into the Fog environment.

2.4.3 Kerberos [16]

Kerberos is a centralized server used to secure untrusted open networks with unsecured machines, where servers cannot authenticate its clients, thus a third party is needed in order to secure the network. Kerberos acts as a ticket issuer, each machine has to identify to the Kerberos server, and then it gets a ticket. That ticket could be used in order to establish secure connections to other machines.

Although this technology is able to secure unsecured networks providing identities to machines, it's not usable on a Fog environment, as it's totally centralized and relies on a ticket database, implying that this system could never scale to a Fog scale.

2.4.4 BCTrust: A decentralized authentication blockchain-based mechanism [17]:

BCTrust proposes a new blockchain model, based on a customized Ethereum[4], that will store each device request in order to perform an exchange of data with another client.

The blockchain will store which devices are trusted by a super node, and then when the device moves to another super node, the supernode checks the blockchain for the newly joined device. If that device was indeed authorized from another super node, then it's valid for him, and requests to the other super node the device symmetric keys to perform a secure data exchange.

In our opinion, the blockchain scalability is the major flaw of this proposal. If we have to store all this information in Ethereum, a blockchain that can store almost anything, the constrained Fog devices will run out of memory, and we won't be able to do nothing to prevent this. In the other hand, this project does not use the whole potential of the blockchain for a totally decentralized authentication, since devices are still forced to perform transactions in order to exchange data securely. An enhancement could be actually use the local blockchain to perform this authentication action without the need of further transactions. This could save a countless and precious amount of memory to Fog devices, memory that is finite on them, as the Fog is not the cloud and its device resources are limited. Apart from this, we have to take into account that symmetric keys are being stored and shared trough the blockchain, and this is a bad practice. The blockchain should never be used to share not static data, as this will make the blockchain grow in data size uncontrollably and will bring scalability issues. There are other technologies already for this purpose.

Another issue is that this proposal does not mention any consensus algorithm at all, and the one used in Ethereum (PoW) is not suitable for this kind of permissioned network. They mention a modification of Ethereum with smart contracts but fails to mention how the consensus validators will be authenticated to other devices. To conclude, in our opinion, if Ethereum got to be modified so much, perhaps Ethereum is not the right base to start with.

2.5 Blockchain Frameworks

In this section, we will overview some of the existing frameworks or projects that could be related or used for our work.

There is one major project that at this moment has 6 sub projects under development and around 6 tools called the Hyperledger project[18] under the Linux foundation. The project goal is to provide the tools needed in order to build blockchains out of the box, abstracting the network low level complexity. The two most relevant Hyperledger projects for our project are Hyperledger Indy or Sovrin[1] (they are not the same project but are highly related as they share the same code base), and Hyperledger Fabric.

Hyperledger Indy[8]

The main goal of Hyperledger Indy is to establish a base for decentralized online identity. The project objective is to provide digital identity to users that could be used in order to use online services.

This is the closest project to our proposal, but it's aimed towards users for the whole net. In the other hand, our proposal is aimed more for Machine-to-Machine (M2M) communications and it's centered on the Fog network architecture.

Hyperledger Fabric[19]

The main goal of this project is to provide a framework where to develop modular business blockchains. It was firstly contributed by IBM and it allows components such as consensus algorithms to be plug-able, or in other words, that they can be changed depending on the environment.

The conclusion after reviewing these frameworks is that they are too much specialized in order to be used in this project. Each framework aims to solve a specific problem. The most related one, Indy, diverges too much from our objective, as it's focused on a distributed digital identity for online services. For this reason, we won't use any major blockchain framework. We prefer to build a prototype of our own in order to test our proposed architecture, but we will closely follow Indy project as it could be, in a future a usable code base for our project.

3 SECURITY CHALLENGES ON THE FOG

In this section we will overview some of the main challenges when applying security to a Fog scenario. First we will start with the mobility issues, later on we will present the centralized security issues, and finally the centralization of data problem.

3.1 Mobility issues

Most security models are static, and are made for static domains or IPs (i.e. Certificates tied to IPs or domains), thus fail to deploy correctly into the Fog. Fog by default is on the IoT devices, and IoT devices are meant to be highly volatile as they can move. Therefore, having a static security model designed for cloud and web applications is not feasible or scalable when applied to Fog.

There are other centralized servers that do not rely on domain names in order to apply security to its clients, since those models are highly centralized. An example would be the reviewed Kerberos, is invalid to apply security to the Fog, as it will create a bottleneck on the Kerberos server.

In conclusion, we require new models that are scalable and at the same time are flexible enough to allow the device mobility without compromising the security, moreover we require new security models that do not rely on a device IP or DNS domain to establish trusted connections with a device.

3.2 Centralized security models

As we have seen there is no distributed security model that can fit in the Fog environment. All major systems are centralized, at some point, and even if we partially decentralize them, for example a distributed CA architecture, they are not flexible enough to fit into the Fog and manage IoT devices security.

In the Fog, devices should be able to move and connect everywhere, we cannot use a security model that does not provide this flexibility, we require a highly distributed model capable of tolerating highly mobile IoT devices but at the same time the model must be scalable, as the amount of connected devices may grow exponentially fast.

3.3 Centralized data

Usually user data is highly centralized, a scattered across various providers, this forces the users to keep track of multitude of accounts and different passwords that usually end in data breaches. This model is not valid for Fog; we can't expect to have tons of accounts for each device as it fails to scale with the billions of devices. Fog requires a unified and distributed architecture to keep all the devices profiles while at the same time being secure, scalable and fault tolerant.

4 APPLYING BLOCKCHAIN TO SECURE A DISTRIBUTED SET OF CLUSTERS

4.1 Objective

The main objective of this project is to provide a distributed Fog session shared among all Fog devices. This session will be used in order to provide security between devices in a Fog scenario, it will enable devices to search and authenticate other devices in the session, establish secure connections between devices or sign data in order to broadcast it to other Fog devices.

To do so, in a high abstraction level, the session will store the devices public keys and it will associate a rule-set with those keys. The rule-set will be a set of actions that are performed on the session for the given keys. For example, a register in the session will be one rule-set, where a device will register into the session, providing its keys and a Fog specific profile. In order to distribute and synchronize this session among all devices we will use blockchain technologies, since blockchain is the right tool to keep an immutable and distributed set of data in form of transactions, where each pair of public keys and rule-set will be a new transaction. The usage of blockchain allows us to have the session distributed across all devices, thus enables a completely distributed and immutable system. The session will contain as many transactions as necessary, and there can be more than one transaction with the same pair of public keys (see illustration 6).

To perform the consensus securely, as already stated on the consensus point 2.2.3, we will require a special set of devices composed by the validators of the consensus protocol, which will be the ones performing the consensus operations over the blockchain.

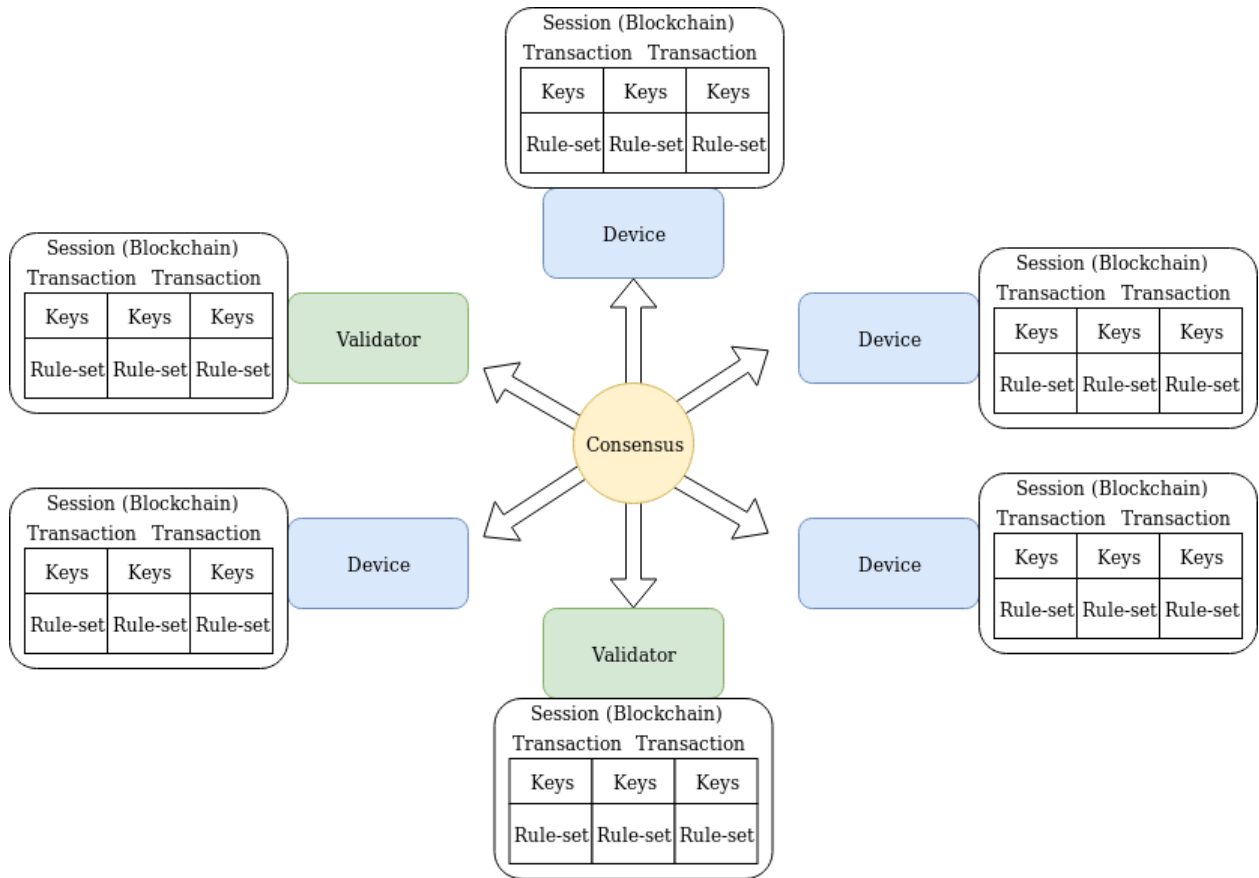


Illustration 6: Abstract objective session

In illustration 6, we have the whole abstract picture of the projects objective, with the session in form of blockchain transactions, distributed across all devices and validators with a consensus protocol.

In the following points, we will introduce the keys, rule-set and consensus used in this project.

Keys

The public keys will be the ECC (Elliptic Curve) type for establishing a secure TLS connection between devices and ED25519 (Elliptic Curve) for signing messages and later on sending them to one or multiple devices. We decided to have two pairs of keys because in the Fog anything can be a device. For example, perhaps there can be a sensor that just wants to broadcast or multi cast data across the network and will never do anything else, then assuming we don't want to keep the data private, just secure, by verifying the data send by the sensor. We can just set a pair of public ED25519 keys for that sensor and use them to sign the data. Then when a device receives the data it can use its local copy of the session to verify that the data is from the expected sensor. In addition most consensus algorithms use ED25519 keys for signing votes, so this key can be used for various purposes depending if the owner is a device or a validator.

In extend, we have chosen Elliptic Curve keys as this type of keys are the ones with the better performance per key size, thus we can use smaller keys with good cryptographical strengths (see illustration 7)[20].

For example, a temperature sensor in a smart city doesn't require data privacy, as anyone can read the temperature, but we require to know that the data is indeed from the expected sensor and not from an attacker's sensor.

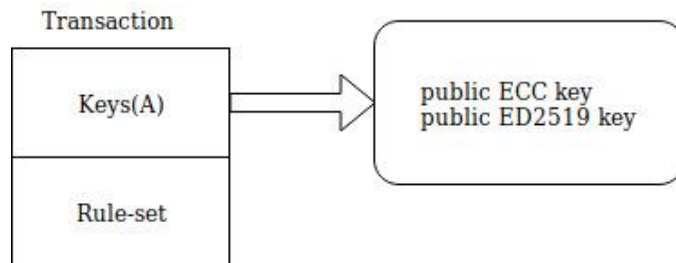


Illustration 7: Public keys in the transaction

Rule-set

The rule-set will be an extensible field that will tell other session members what a device is, and what it is allowed to do inside the session. Thus the rule-set is a set of rules applied over a public key in form of a transaction. There can be infinite rule-sets applied over the same keys. Since we are going to use a blockchain, we cannot erase or update any rule-set that has been committed, we can only commit a new rule-set, and the devices should trust the most recent transaction.

The rule-set should at least contain the Fog profile of the devices (rule-set device) or validators (rule-set validator) in the session, with that information being provided at registration time. If there was a specification standard for Fog devices, then the rule-set device or the rule-set validator should at least contain that standard. But since there is no standard at all, we propose an example of what those profiles could be. We propose to add the devices capabilities inside the Fog area, this means to specify what a device is and what it can do, for example for a device e.g.(RaspberryPy) with a temperature sensor, we will specify that we have one device with its computation capabilities, such as CPU,RAM,OS and that this device has a temperature sensor. For the sensor we can specify the type of sensor and sensor attributes that could be customizable. We can also specify the expected data range of the sensor data, in order to help the network in identifying wrong sensor data. The computation capabilities and sensors could be used later on when executing a service over the Fog infrastructure, while the sensor description and data range would be used to verify the data received from the sensor. The rule-set device or rule-set validator will only be committed once on the session at registration time, and the devices or validators will not be able to further modify it. This is done to guarantee the profile security on the session, as no attacker will be able to alter a device or validator profile even obtaining the device or validator private keys (see illustration 8).

Since Fog is a M2M environment, we understand that profiles should be static, and for this project they will stay static. But if the corresponding environment requires the profile to be updated, it could be accepted assuming the security risk of doing so (i.e. Attackers could update profiles if gaining access to a device private keys). In other more complex Fog scenarios, the rule-set could be used to add extra attributes to a device. For example, we could add rules of how our device can interact with our system and what it's allowed to do. But the rule-set should only be used to add "static" information about a device, and it should never be used to add dynamic information that changes frequently. Doing so will overload the session and the devices, any dynamic data should be shared on secure channels using the public keys provided by the session.

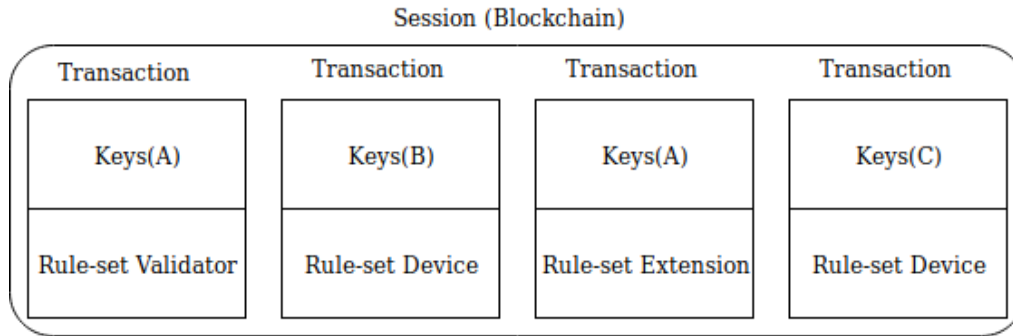


Illustration 8: Rule-set examples

Since we can register different type of profiles, the rule-set will be different depending on the type of device registered, thus when registering a device, the rule-set will be the rule-set device, and when registering a validator, it will be the rule-set validator and so on. Both are a rule-sets representing a profile, but the profile will have a different structure as we will represent different kinds of devices. As stated, we will have a rule-set validator that can be used to add new validators into the session. Moreover, it will be the first transaction of the whole session, informing of the first validator of the blockchain consensus algorithm. To add new devices/validators, a device with a validator profile will have to issue a new transaction informing of the newly added devices/validators, and sign that transaction with its private keys.

Other rule-set use cases would be the blacklist, since we can't remove any data of the session, we need a mechanism to inform the session devices of a malicious device or validator. This mechanism is the rule-set (blacklist). This rule-set will automatically tell other devices that a set of keys is no longer trusted. In the case of blacklisting a validator, or a device, a timestamp may be specified when issuing the blacklist, providing a point from where any action wouldn't be trusted. This is an extra information to help the network in identifying malicious activity from compromised devices or validators.

And finally, the last use case for the rule-set, the gateway, that is intended to represent a gateway device, a special case of device, common in Fog and IoT environments and that we will explain in detail in the following components part 4.2. The concept is to have a device, with registered sub devices. Therefore, the sub devices security is outside the session control, the gateway will have to be trusted by the Fog devices.

It's important to point out that the rule-set can be totally customized, and new rule-sets can be added as extensions in order to modify the purpose of the session and adapt it to the required environment.

Consensus

Consensus algorithms are already a well explored field, we do not pretend to create a new consensus algorithm, but to use the various available consensus algorithms that are already out there and that we have reviewed on the state of the art. The only thing we require is a permissioned consensus algorithm that allows the use of validator devices.

That said, most suitable consensus algorithm would be RBFT, Zyzyva or the RBFT modification of the project Indy, being RBFT the best one. The consensus algorithm must be permissioned, so we can use a set of consortium owned validators to control the Fog session and should offer the best possible performance. We propose that the project implements more than one consensus algorithm and then, when doing the set-up of the session, we could choose which algorithm will fit better into our environment.

On the prototype code implementation, it is infeasible to implement or reuse one of these consensus algorithms as it would take too much time due to the integration complexity. For this reason, and since the consensus algorithm just proves the efficiency when committing and distributing blocks, we have chosen to use a much more basic consensus algorithm based on rounds and votes. On each round, a validator proposes a block and is submitted for voting. If all the validators approve it, then it is committed. The devices do not take part in this process and are updated when a new block is committed.

This basic algorithm approach is not Byzantine fault tolerant and can easily deadlock, but since it's just intended to be there for a prototype version where device additions and in extend all actions performed over the session will be controlled by us, this should not be an issue.

Standardization

We do not pretend to create a Fog device profile standard, as this should be discussed with major vendors thus it's out of scope of this project. First of all, we want to prove that the concept works, for this we have created our own fictional standard, but in a commercial version, this should change to a unified profile.

4.2 Components

In this section, first we will define what each session conceptual component is in our architecture, and later we will do an overview of each component characteristics in form of table and graphic.

4.2.1 Secure Edge Node (SEN)

SENs, refereed as validators on the blockchain consensus algorithms, are the controllers of the consensus and are responsible of keeping the security and integrity of the session. SENs can be added into the session at any moment, and the first rule-set performed over the session, must be the one adding the first SEN, we refer to as SEN zero (SEN0).

When a SEN is added, a rule-set specifying the new SEN allowed actions is added with the SEN keys. This rule-set tells others what the SEN is allowed to do. For instance, if it's allowed to add new SENs or new devices to the session, and to blacklist SENs or devices from the session.

Apart from that, SENs can better perform the bootstrap of network peers, as their locations should be more static than the ones from devices, and should keep a Distributed Hash Table (DHT) of network devices last known IP (i.e. network state and peer discovery).

4.2.2 Device

A device can be anything that meets the requirements in order to run the framework and keep the local copy of the session's blockchain.

By default, a device is registered in the session with its capabilities such as CPU, RAM, GPU, OS, and if it's under power or battery. Devices can add up to n sensors in their profile, specifying trough an extension each sensor type and characteristics and since devices are part from the session and hold a copy of it, they can use to session to broadcast signed sensor data, or establish TLS connections to other devices/SENs in the session, along with authenticating the data received.

4.2.3 Gateway

A gateway represents a set of devices that do not have direct connection to the session, but want to share its data with the session devices trough the gateway. It can register a set of devices, each one with its pair of keys. The devices behind the gateway will be able to establish secure connections and broadcast data to the whole session, but the session won't be able to reach them directly. Devices will refer to the gateway in order to reach those devices if necessary. In addition, security outside the gateway can't be guaranteed, so the session devices will have to trust the gateway provided data.

4.2.4 Simple Device

A simple device is a special case of a device that it's part of the session but does not keep a local copy of the blockchain thus it can send data securely but cannot validate any incoming connection or data against the session. The objective of this component is to represent devices more focused on broadcasting sensor data than on receiving connections, thus they would be part of the session, but they won't hold a copy of it.

4.2.5 Anonymous device

An anonymous device can be anything that meets the requirements to run the project session application but doesn't want to be registered in the session.

The anonymous device it's able to validate the data provided by other devices and authenticate them but no one can validate it through the session since it's not registered. This can be useful when performing the authentication steps on an external service to the session, then the session can be used in order to identify the service providers to the anonymous clients for the session.

4.2.6 Resume of component characteristics

In the following table, table 2, there is a comparison between the architectural components and their characteristics inside the Fog session. Later on there is an illustration with the same comparison between devices characteristics.

Characteristics	SEN	Device	Gateway	Simple device	Anonymous device
Has a copy of the blockchain	Yes	Yes	Yes	No	Yes
Can add new devices or SENs	Yes	No	No	No	No
Can validate TLS connections	Yes	Yes	Yes	No	Yes
Can validate data signatures	Yes	Yes	Yes	No	Yes
Can be validated on TLS connections	Yes	Yes	Yes	Yes	No
Can be validated when signing data	Yes	Yes	Yes	Yes	No

Table2: comparasion of each component characteristics

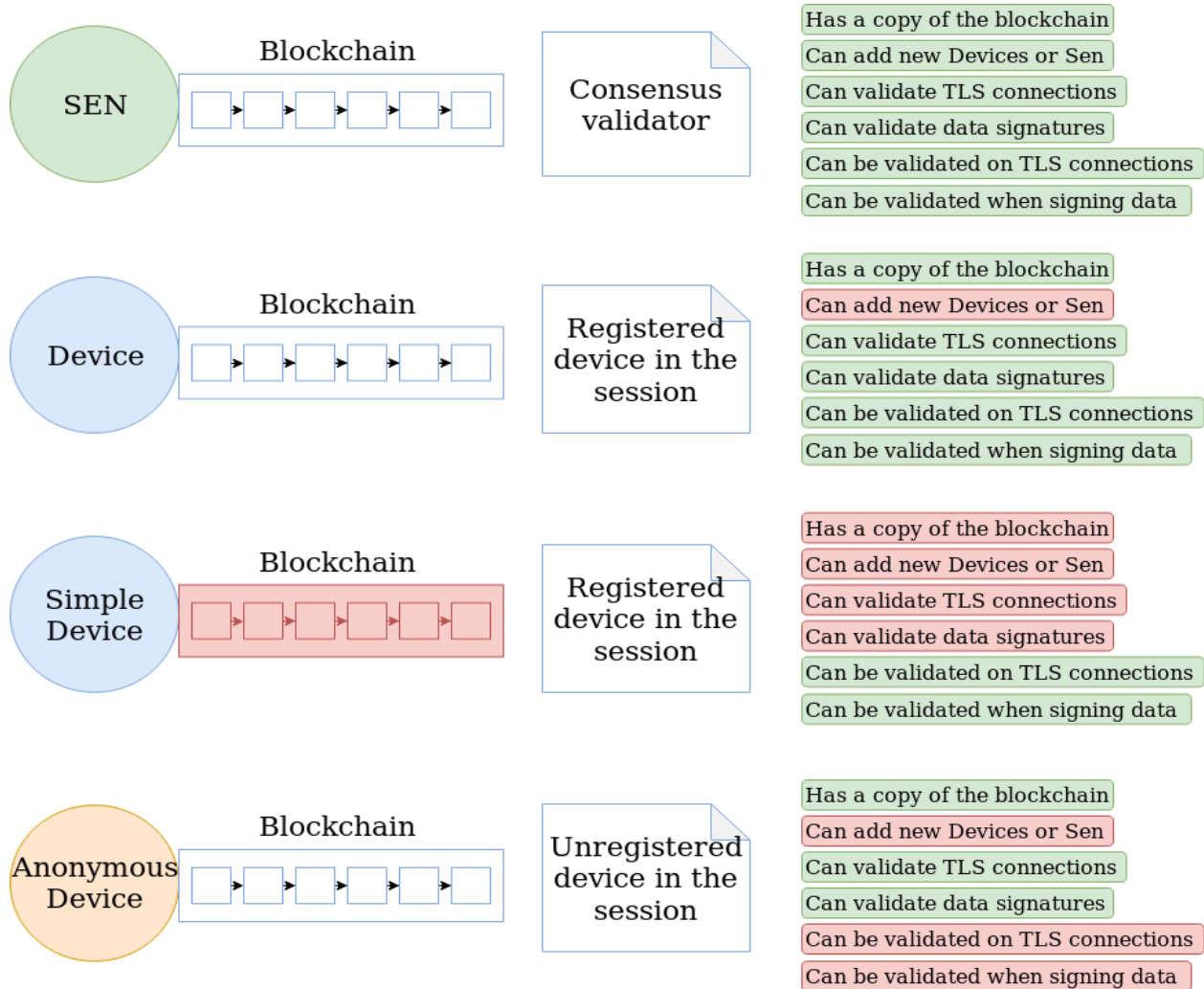


Illustration 9: Graphical comparison of each component characteristics

4.3 Proposed architecture

4.3.1 Network

The proposed network for this project, since it's Fog related, it's a fully peer to peer network (see illustration 10), where all the nodes must be reachable. If any node cannot be reached, it will fail to receive consensus votes from SENs or connections and data from other nodes. SENs can be used as tracker devices as done in torrent networks[21], with the addition of a new rule-set in the session.

Then SENs would be used to keep a DHT of the network peers as done in torrent[21], in order to allow other devices to discover all the peers. A DHT is a distributed map of the network that stores devices IPs and performs peer discovery, one example would be[19].

This will solve the problem of device lookup, since we can search the session for devices. However, since the session doesn't (and should not) store any device IP, devices can't find other devices in the network. Therefore, we propose to use DHT for device discovery. In resume, devices could search for other devices in the session and then query SENs or trackers with the desired device public key, in order to obtain the last known target device location from the DHT network. The best solution for this would be the Kademia DHT[22], as torrent has already shown its good performance when performing network discovery on a peer to peer network (see illustration 10).

P2P network

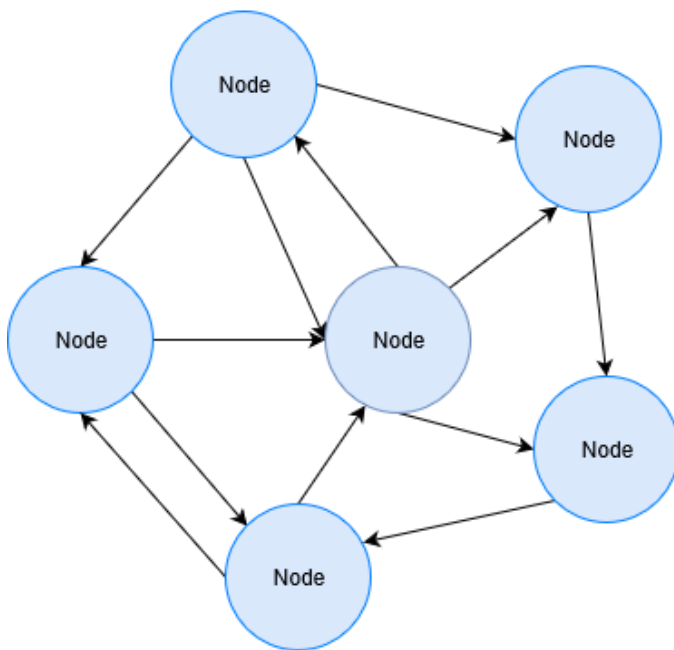


Illustration 10: P2P network example

4.3.2 Blockchain: Blocks

In this chapter we will define the block, the transaction and the various session component rule-sets inside the transactions.

Blockchain: Block structure diagram

In the following figure (illustration 11), we can observe the main hierarchy of the data structures. First we have the block that holds the transaction, containing a set of public keys and the rule-set.

This rule-set can take various forms, seen on the first row {Device, Sen, Blacklist, Gateway} and those data objects can be extended further more with the second and last row of structures {Data, SensorExt, SenExt, Complex,Device}. Some of the data fields are marked as optional, meaning that are not required to be filled when used in a rule-set (see illustration 11).

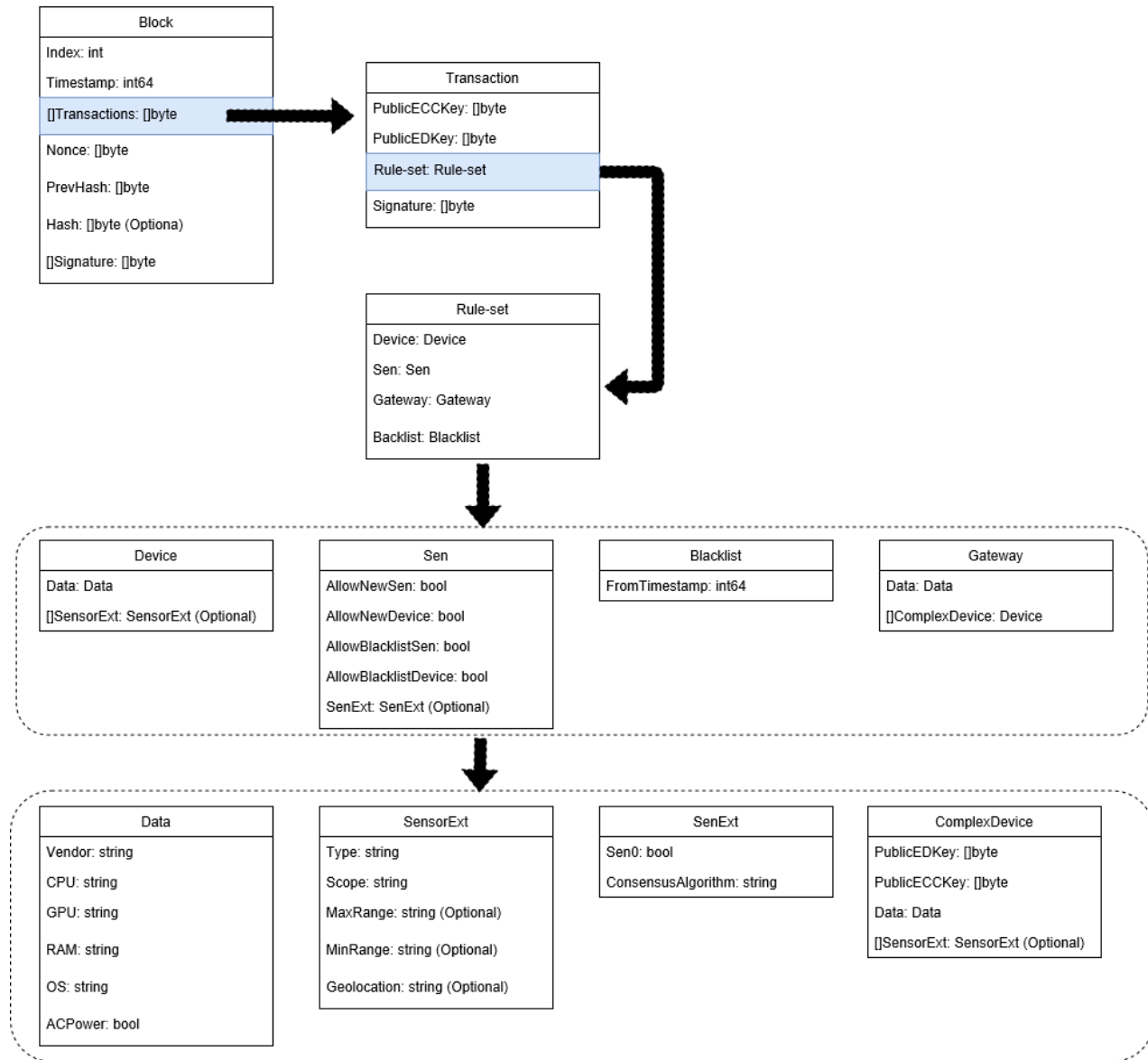


Illustration 11: Hierarchical representation of the blockchain, blocks, transactions and possible rule-sets.

The rule-set will be encoded with Protobuf or Protocol Buffers[23]. Protobuf is a serialization tool that is platform and language neutral and helps encode and decode data structures in a highly efficient form. Although it's usually used for network communications as its name suggests, it can also be used to encapsulate data and store it, so when the data is distributed to multiple users it can be easily reversed. This is important as it effects the structure of the block, which must be designed in order to accommodate the serialized rule-set data.

Block (Blockchain)

The block is the highest component in the hierarchy, is the object that holds all the other objects, and is the one that forms the 'block chain'.

- **Index:** The order of the block in the blockchain.
- **Timestamp:** The time when the block was created.
- **Transactions:** The payload of the block, contains the transaction with the keys and rule-set, that represents the session operations.
- **Nonce:** A random array of bytes to strength the cryptophysical hash of the block by adding entropy.
- **PrevHash:** The hash of the previous block in the blockchain.
- **Hash:** The hash of the block, this is here for testing purposes on the prototype.
- **Signature:** The signature or signatures of the block issuer, added by the consensus algorithm.

Transaction (Block)

Represents an action trough the rule-set over the given public keys on the session. The transaction is part of the block.

- **PublicECCKey:** The publicECC key of the device/SEN associated with the rule-set.
- **PublicEDKey:** The public ED25519 key of the device/SEN associated with the rule-set.
- **Rule-set:** Represents all the possible models of a rule-set.
- **Signature:** The signature or signatures of the transaction creator. This should be signed by SENs and it's done in the case of adding multiple transactions on the same block in the consensus phase in order to authenticate the transaction proposer.

Rule-set (Transaction)

The rule-set has all the possible forms of a rule-set, it's a Protobuf implementation requirement, in order to later on codify and decodify the rule-set data. The rule-set is part of the transaction.

- **Device:** The device representation of the Protobuf device object.
- **Sen:** The SEN representation of the Protobuf device object.
- **Gateway:** The gateway representation of the Protobuf device object.
- **Blacklist:** The blacklist representation of the Protobuf device object.

Device (Rule-set)

- **Data:** A data object representing all the device characteristics.
- **SensorExt:** An array of objects representing the sensors attached to a device.

SEN (Rule-set)

- **AllowNewSen:** Represents either if the SEN is allowed to add new SENs or not.
- **AllowNewDevice:** Represents either if the SEN is allowed to add new devices or not.
- **AllowBlacklistSen:** Represents either if the SEN is allowed to blacklist SENs or not.
- **AllowBlacklistDevice:** Represents either if the SEN is allowed to blacklist devices or not.
- **SenExt:** A SEN extension object representing the SEN0.

Blacklist (Rule-set)

- **FromTimestamp:** The timestamp from where the blacklist is effective.

Gateway (Rule-set)

- **Data:** A data object representing all the device characteristics.
- **ComplexDevice:** An array of objects representing a device with the addition of its public keys.

Data (Rule-set)

- **Vendor:** The vendor of the device
- **CPU:** The CPU of the device
- **GPU:** The GPU of the device
- **RAM:** The ram of the device
- **OS:** The operating system of the device
- **ACPower:** If the device is on Ac power this will be set to true, else way will be set to false.

SensorExt (Rule-set)

- **Type:** The type of the sensor, since there is no standardization done here, the type will be an approximation of what the real type under a standard would be.
- **Scope:** The scope represents extra attributes to the sensor.
- **MaxRange:** The maximum expected range of the sensor data.
- **MinRange:** The minimum expected range of the sensor data.
- **Geolocation:** The GPS location of the sensor, but only in case of static sensors in buildings, etc.

SenExt (Rule-set)

- **Sen0:** Set to true to represent the SEN0.
- **ConsensusAlgorithm:** The consensus algorithm chosen by the SEN0 for the session.

ComplexDevice (Rule-set)

- **PublicECCKey:** The publicECC key of the device/SEN associated with the rule-set.
- **PublicEDKey:** The public ED25519 key of the device/SEN associated with the rule-set.
- **Data:** A data object representing all the device characteristics.
- **SensorExt:** An array of objects representing the sensors attached to a device.

4.3.3 Blockchain: Consensus

As we have already covered in the objectives section, the consensus that will be applied on the prototype will be significantly simpler than the designated consensus algorithm (RBFT). It's not needed to implement such a complex algorithm when in fact consensus is a tool needed for the prototype and the testing and validation of the use cases, but it's not needed for the architecture, as the consensus is already proven to be effective on the RBFT paper[6].

SEN Consensus Protocol

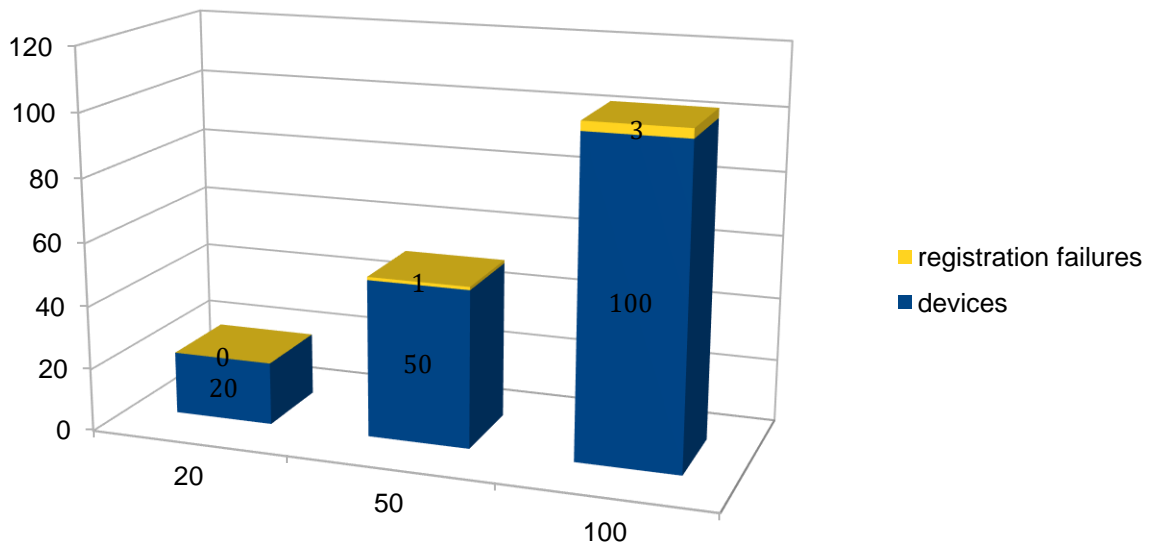
Our consensus algorithm will be more than enough for all the desired testing scenarios of the prototype. Although it won't be fault tolerant and won't have a good scalability.

The algorithm won't include any device in the consensus process, and devices will be updated with new blocks periodically by SENs or by forcing an update sending a query to any SEN, with their last known block index. Finally, we won't allow more than one transaction per block.

Following this, only the SENs will be participating in the process of consensus, and all the votes will be sent between SENs. In the first step, at any time, a SEN may propose a new block for voting in a voting round. If the SEN is already in a voting round, the proposal will be queued until the actual round finishes. When a SEN receives a proposal, if it's not engaged on any other voting round, it will lock into the round. Then, it will validate the block and broadcast a vote to all the other SENs, accepting or denying the block. When a SEN receives a vote it's stored, the round finishes when a SEN has received enough votes to validate the proposed block. For us, this number will be equivalent to the number of validators present on the session at the start of the round (thus if a validator fails mid round the voting will block until a timeout is reached and it starts again). When a SEN receives all the votes, then the block is said to be committed (All the SENs participating in the consensus must have the same copy of the blockchain at this point) and the round is finished freeing the SENs for a new round.

Basic Test

We have performed a test of this protocol to validate if the protocol is valid to run on virtual scenarios with a considerable amount of devices. In the following graphic (graphic 1) we can observe the results of the test that would launch a new device every five seconds, the new device will try to perform a login in order to test the consensus phase. The tests were done on the same environment as the final testing scenario, and all the logins were performed on a 4 SEN network.



Graphic 1: results of SEN consensus protocol testing number of devices vs registration failures.

In the results, we can observe that the amount of registration failures due to consensus locks are really small, as we have 0 locks for 20 devices, 1 lock for 50 devices and 3 locks for 100. The locks come from the inconsistency between ended rounds and open rounds that can momentarily happen between SENs. this would be a huge issue on a large scale with billions of devices but not on our testing environment or prototype scenario.

In conclusion we can tolerate this amount of registration failures in the final testing of the prototype. For instance, the failures are on the voting rounds, and all the devices and SENs keep a consistent copy of the blockchain, thus the failures are only from devices failing to register. However, this never creates any inconsistency in the resulting session blockchain, it simply forces devices to retry the process.

Consensus Flowchart

Following we illustrate the flowchart of the SEN consensus protocol.

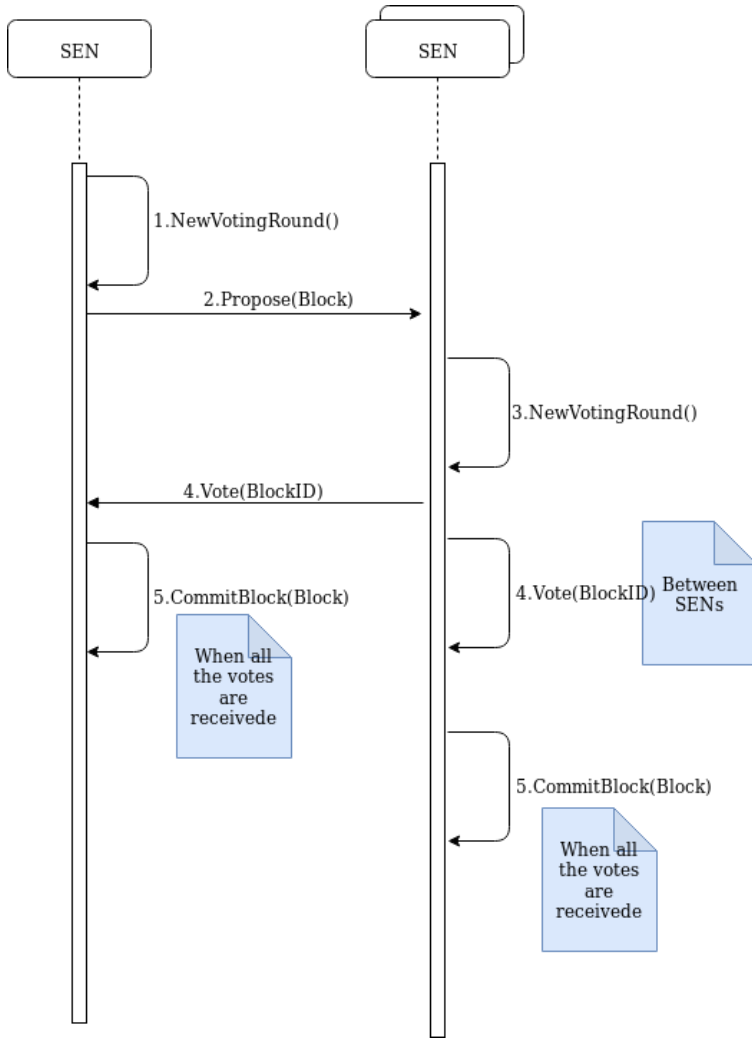


Illustration 12: SEN consensus protocol flowchart

In the SEN consensus protocol flowchart (see illustration 12) we have the following steps: (1) on the proposer SEN a new round starts. (2) A block is proposed by the proposer SEN and broadcasted to all the other SENs. (3) Other SENs start a new voting round. (4) SENs broadcast their votes for the given block in the actual round. (5) Once a SEN receives as many votes as other SENs present in the round the block is considered committed and the voting round ends.

4.3.4 Webinterface

In order to make any demo more visual, we will implement a web interface for the devices and SENS. the web interface will have three main tabs, the dashboard, the device or self-information, and the blockchain.

In illustration 13, the dashboard will present an overview of the status of the session, displaying the number of SENS and devices, the total number of committed blocks in the blockchain session and the bytes used by those blocks.

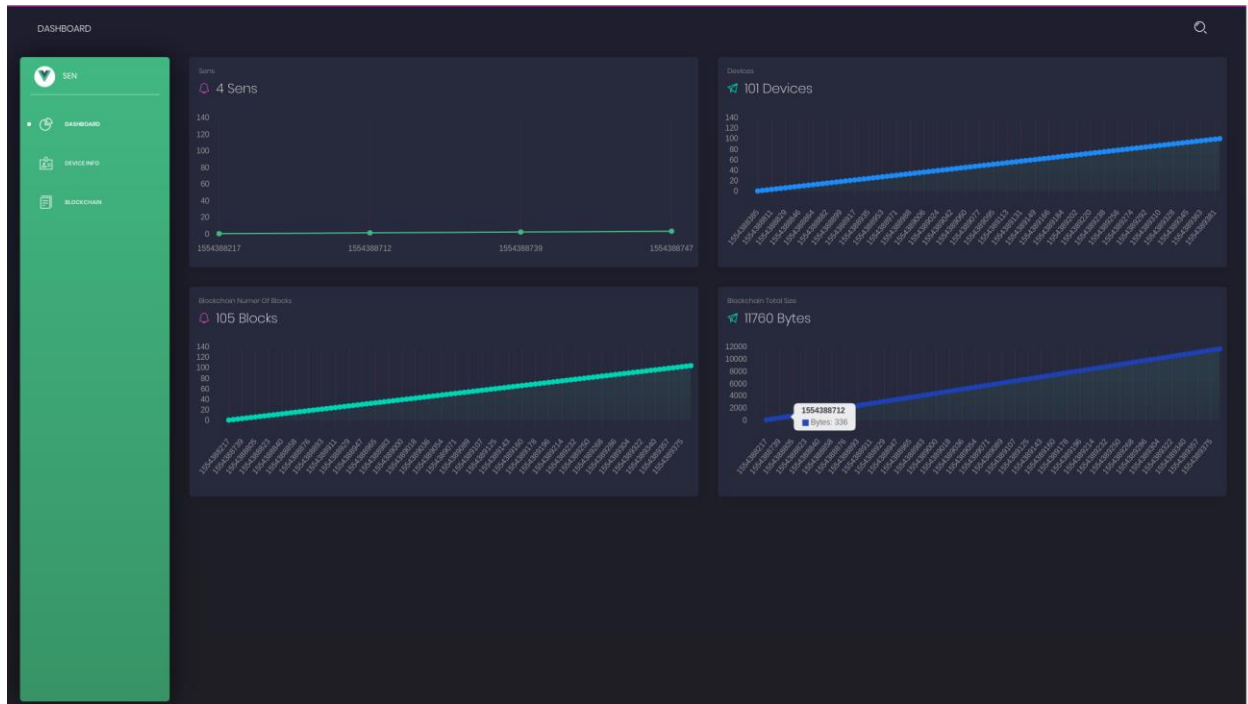


Illustration 13: Web interface dashboard

In illustration 14, the second tab in the menu will be the device info and it will contain the rule-set (profile) of the device that we have registered into the session.



Illustration 14: Web interface device info

Finally, in illustration 15, the third tab will display the whole session blockchain into the browser, we will be able to see the devices and SENs, and its associated rule-sets.



Illustration 15: Web interface blockchain

This web interface will be available at each device and SEN, since it's just for visualization purposes and no action can be taken on the session, it just provides a feedback, it will be open to anyone capable of reaching the URL.

4.3.5 Architecture integration

In illustration 16, we can see the resulting picture representing the whole session, in a high level. We have SENs, Devices and a Gateway. All of them have sensors and the Gateway has a subset of registered devices. All of these network devices are part of our Fog session, and can securely communicate and exchange data. Thus the Fog network has been successfully.

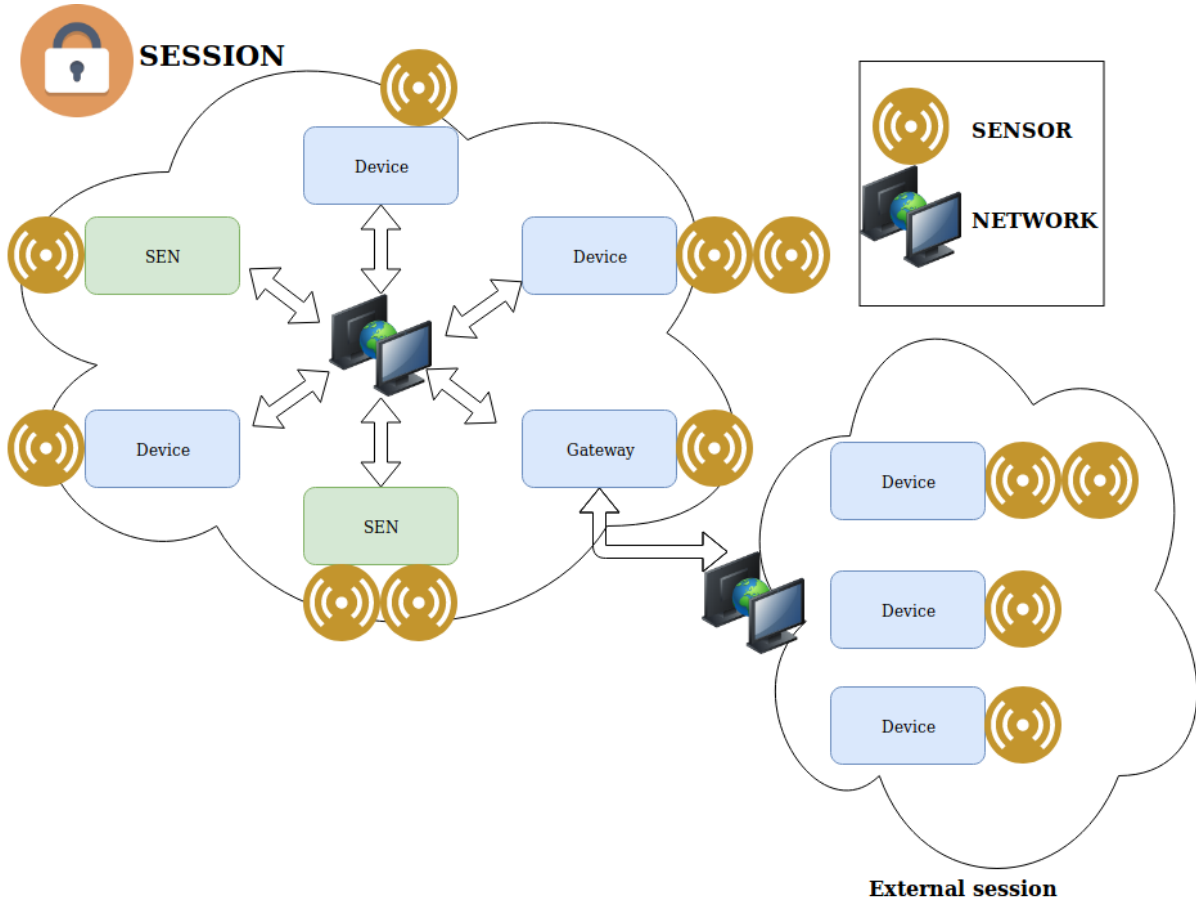


Illustration 16: Complete diagram of the whole architecture

Then if we go back to the first objective diagram (illustration 6), we could slightly modify it in order to better represent what the session actually is in terms of our internal architecture. Our session can contain SENs and various forms of devices such as normal devices or gateways, all of them are registered in the Fog session through the rule-set, and each device is able to verify all the other Fog session participants locally (see illustration 17).

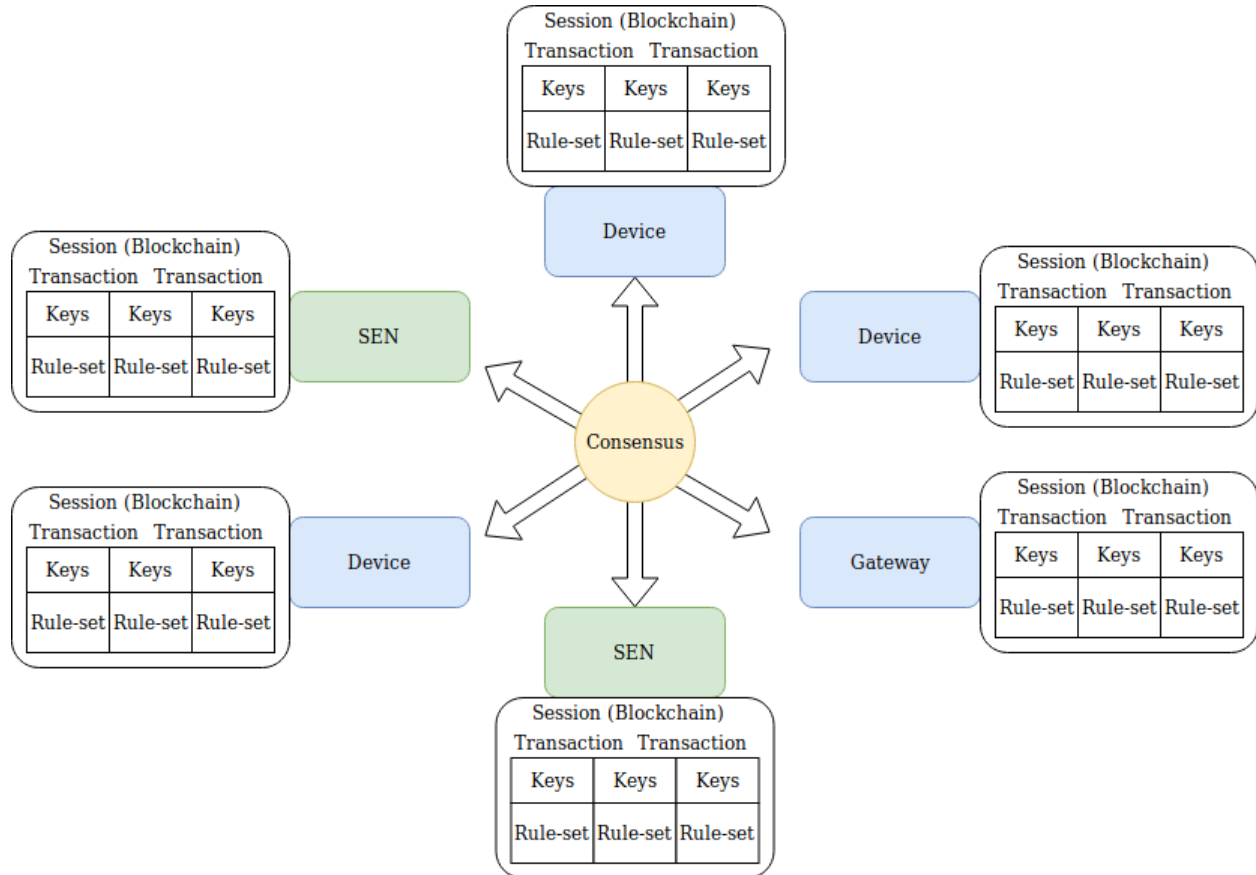


Illustration 17: Update of the objective diagram

4.4 Functionality

The project main component is a session stored in a blockchain and composed by blocks. The blocks in the session contain transactions that link the devices public keys with a rule-set. This rule-set is the one storing all the session actions performed over the public keys that identify a device. This architecture allows to have all the devices registered in the rule-set with a special rule-set of registration. However, the rule-set could contain much more information than that, for the devices it's simply the registration of that device, but the registration of a SEN contains completely different information. An important key aspect to remark is that the session it's distributed across all peers, everyone can validate the session and the other IoT devices against its own session copy without the need of a third party.

The data stored in the session can be customized, thus this architecture can fit in almost any Fog deployment and adapt to its network needs, or could be even used to apply distributed security to other areas that require this kind of distributed model.

4.4.1 First key distribution.

When creating the session for the first time, the first thing to do is the registration of a SEN. This will be the first SEN on the session and it's called SEN0. The creation of the SEN0 contain a set of keys, from these we will create the first transaction, the first block, and the blockchain. The first transaction will have a rule-set telling the others that the transaction is a SEN0, making the session identifiable by any device that wants to join.

The SEN0 public keys are the ones used to identify the session when joining, since those are the ones used to create and sign the first block of the session's blockchain. Thus these are the most critical keys of the infrastructure, we recommend to add other SENs after the SEN0 and then keep the SEN0 offline to harden its keys security.

When a device wants to join the session for the first time, it must know an IP where to bootstrap and get the session blockchain. Later, on this session blockchain, it can be verified with the SEN0 public key, that provides the whole chain of trust in the session. Once the session blockchain has been verified, the device will be able to know all the validators and perform a registration.

In conclusion, devices libraries had to come preinstalled with the SEN0 public key in order to verify the session for the first time.

4.4.2 SEN0

Overview

The SEN0 is the first SEN in the session, and has all the session privileges, it can add any device or any SEN and it can also blacklist them if necessary, and since it's the first one it cannot be removed or blacklisted.

SEN0 Rule-set example



Device Info

SEN

Device ECC public key:
LS0tLSiCRudJTtBQVUJMSUMgSOVZLS0tLS0kTUZrd0V3WUhlbJpJemowQ0FRWUilbJpJemowREFRYORRZ0FFSWVYZ0k5U3RIZUt5RIQvZuhsOUpWYjcxZEtSegoxbUpkWDN3bzUkNBWjcvNmFmNlkrVpMV2wxT0UHRDjhmZRCRjBPdVevdnFk1HRhOG0zMVJ6OTJ3PT0kLS0tLSiFtkQgUfVcTEIDIEtFWS0tLS0tCg==

Device ED25519 public key:
4n/8UfGko0R+yIPQpocPikoHeZZiBcqX4OZGIOrGnA=

Allow New Devices: true

Allow New Sens: true

Allow Blacklist Devices: true

Allow Blacklist Sens: true

Consensus algorithm: debug

Is Sen 0: true

Illustration 18: SEN0 Rule-set

In illustration 18 we have the transaction belonging to a SEN0. First we have the two public keys of the SEN0, the ECC key and the ED25519 key. After that, we have the rule-set representing the SEN registration, which contains the rules about what the new SEN can do. These rules are: Allow New Devices, Allow New SENS, Allow Blacklist Devices, Allow Blacklist SENS, the SEN extension field that contains the consensus algorithm to be used in the session, and finally a flag that represents the status of SEN0.

4.4.3 Sens

Overview

The SEN can be customized when being registered in the session in order to perform differently according to the desired requirements. This customization is done through the rule-set and is explained forward in this chapter on the Rule-set example.

The customization allows the SEN to perform actions over the session. We can choose if we want to allow the SEN to add new devices or blacklist them, and we can choose if we want to add new devices or blacklist them.

SEN Rule-set example

Device Info

SEN

Device ECC public key:
LS0tLS1CRUdJTiBQVUJMSUMgSOVZLS0tLS0KTUZrd0V3WUhlbJpJemowQ0FRWUllbJpJemowREFRY0RRZ0FFNGt5VW9c0NlI4Tzc2TXhXalByR0xhMTMhNnR4TAoySihQaUlhUmgyUmJNHFmTSttUmovVIB
DdmVvmZfBSWk4WDZWTihQZFZ4dkwvZGpKtKNGcGNcUUNBPT0KLS0tLS1FTkQgUUVCTEIEFWS0tLS0tCG==

Device ED25519 public key:
mgilRvid0fKd3uLAXWfdqZk+FJa+8ISwMIU8meWZ9o=

Allow New Devices: true

Allow New Sens: true

Allow Blacklist Devices: true

Allow Blacklist Sens: true

Illustration 19: SEN Rule-set

In illustration 19, we have the transaction belonging to a SEN, first we have the two public keys of the SEN, the ECC key and the ED25519 key. After that, we have the rule-set representing the SEN registration, which contains the rules about what the new SEN can do. These rules are: Allow New Devices, Allow New SENS, Allow Blacklist Devices, Allow Blacklist SENS. In fact, this rule-set is similar to the SEN0 rule-set, and its only differentiated by the fact that it misses the SEN extensions with the consensus algorithm and the sen0 flag.

Registration and consensus flowcharts for SEN devices

Here we will explain the two approaches for consensus and its flowcharts.

Simple join request:

In the join request of a SEN, the devices will **not** participate in the consensus, they will be passive and they will receive updates from SENS in order to keep the session updated, in any case they will be still capable of verifying the session blockchain (see illustration 20).

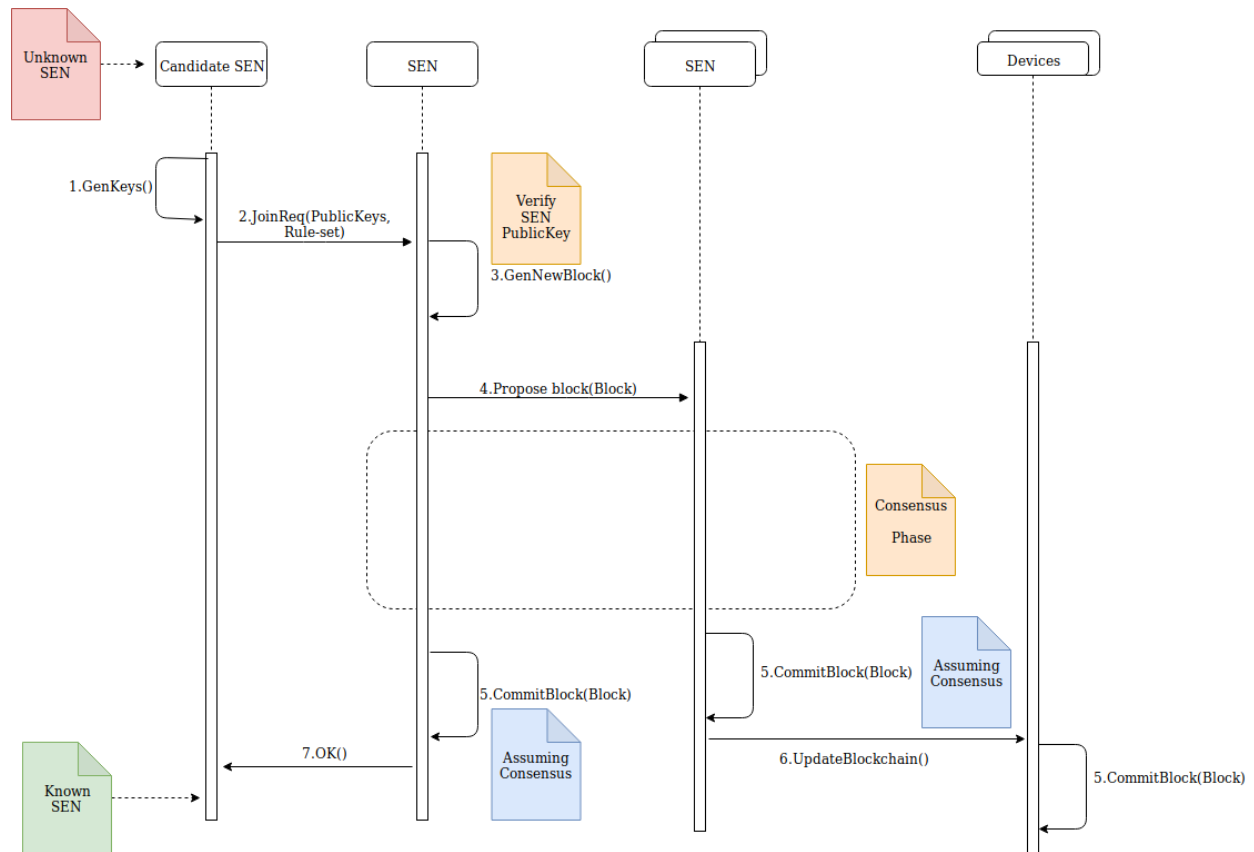
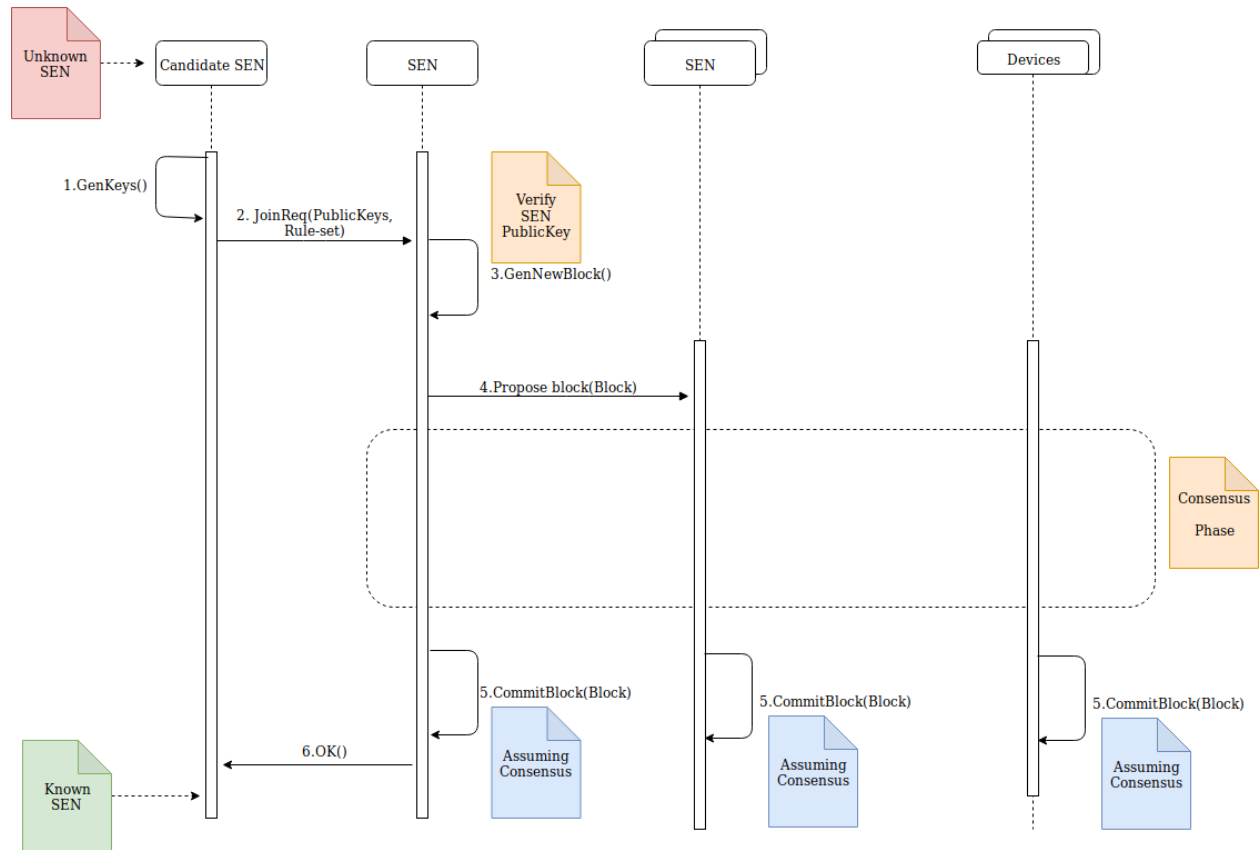


Illustration 20: Simple join request of a SEN using the SEN consensus protocol

In illustration 20, we have the flowchart of this request. The flowchart goes as following: (1) A new set of keys is generated in the SEN. (2) The candidate SEN sends a SEN registration message to the target SEN with its public keys and desired rule-set. (3) The candidate SEN public keys and rule-set are verified against the authorized public keys and allowed rule-set for the candidate SEN. If valid, a new block is generated with the candidate SEN transaction, containing the public keys and rule-set for the SEN registration. (4) The Newly created block is proposed to other SENs in order to start the consensus phase distributing it among all SENs. (Consensus) we won't cover the consensus flow in this diagram, as we will assume that it has gone correctly and we have consensus for the proposed block. (5) The block is committed, from this point the candidate SEN is a SEN and valid in the session. (6) The session is updated on the devices, as they are out of the consensus phase and do not know the newly added block. (7) An OK message is returned to the candidate SEN informing that the registration has been performed correctly.

Complex join request

In this join request of a SEN, the devices will participate in the consensus, they will be active and they will receive consensus messages from SENs periodically (see illustration 21).



Illustration

21: Complex join request of a SEN using an advanced consensus algorithm


In illustration 21, we have the flowchart of this request. The flowchart goes as following: (1) A new set of keys is generated in the SEN. (2) The candidate SEN sends a SEN registration message to the target SEN with its public keys and desired rule-set. (3) The candidate SEN public keys and rule-set are verified against the authorized public keys and allowed rule-set for the candidate SEN. If valid, a new block is generated with the candidate SEN transaction, containing the public keys and rule-set for the SEN registration. (4) The Newly created block is proposed to other SENs in order to start the consensus phase distributing it among all SENs and devices. (Consensus) We won't cover the consensus flow in this diagram, as we will assume that it has gone correctly and we have consensus for the proposed block. (5) The block is committed, from this point the candidate SEN is a SEN and valid in the session. (6) An OK message is returned to the candidate SEN informing that the registration has been performed correctly.

4.4.4 Devices

Overview

As the Fog paradigm states, devices can be almost anything. Thus the registration process of a device is highly extensible, as we will see in the demo rule-set, we can register the device capabilities, and the sensors that it has. These sensors are customizable too. All this information is also registered through the rule-set that specifies a device.

Device Rule-set example



The screenshot displays a 'Device Info' section with the following details:

- Device ECC public key:** LS0tLSiCRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUZrd0V3WUhlblpJemowQ0FRWUllblpJemowREFRY0RRZ0FFb24yVViZTitqdURKaWJZMmNmNitQOURubUhZYgpFQ2tVbVjQ2SUIUWtzRFAxbFNMT3hwckxTRjZHaUFGFzN0s3RVUrMW53ZDBxYU5sM2NKSElCcDBnPTOKLS0tLSiFTrOgUjFVCtEDIEtFWS0tLS0tCg==
- Device ED25519 public key:** ed8lQ0iWnDdubJ0gZL3x9AOENDZsaJJokzUWLxjOI=
- Device Vendor:** MSI
- Device CPU:** Intel
- Device GPU:** NVIDIA
- Device RAM:** 16GB
- Device OS:** Linux
- Device acpower:** false

Illustration 22: Device Data Rule-set

Here we have the rule-set that represents the sensors in the devices

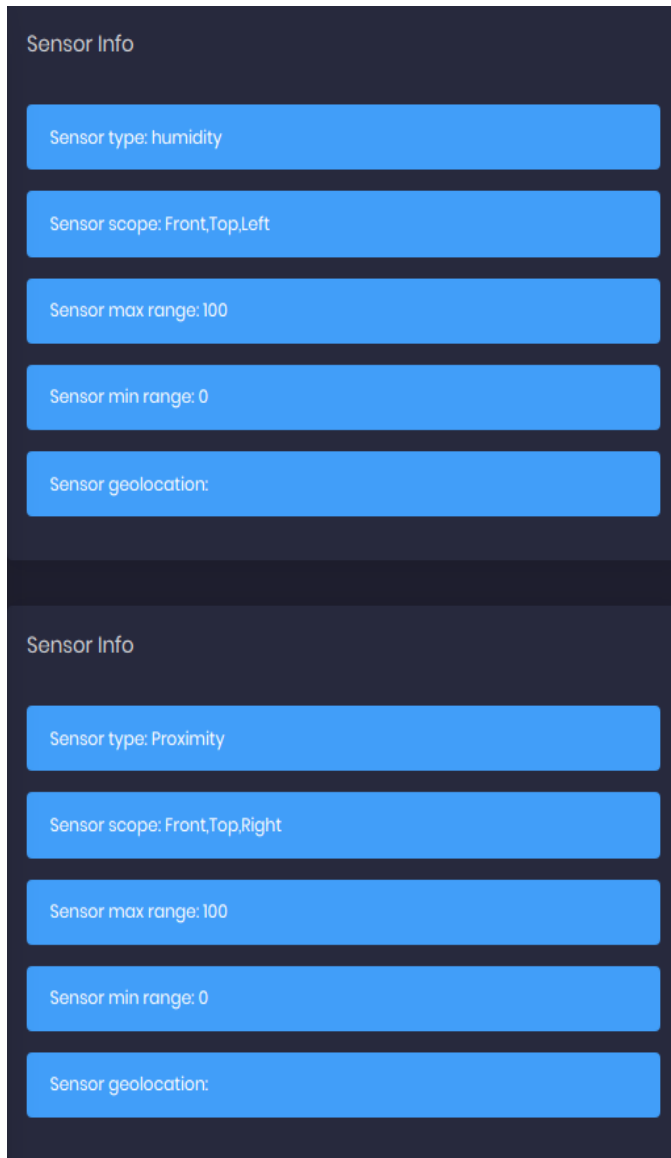


Illustration 23: Device SensorExt Rule-set

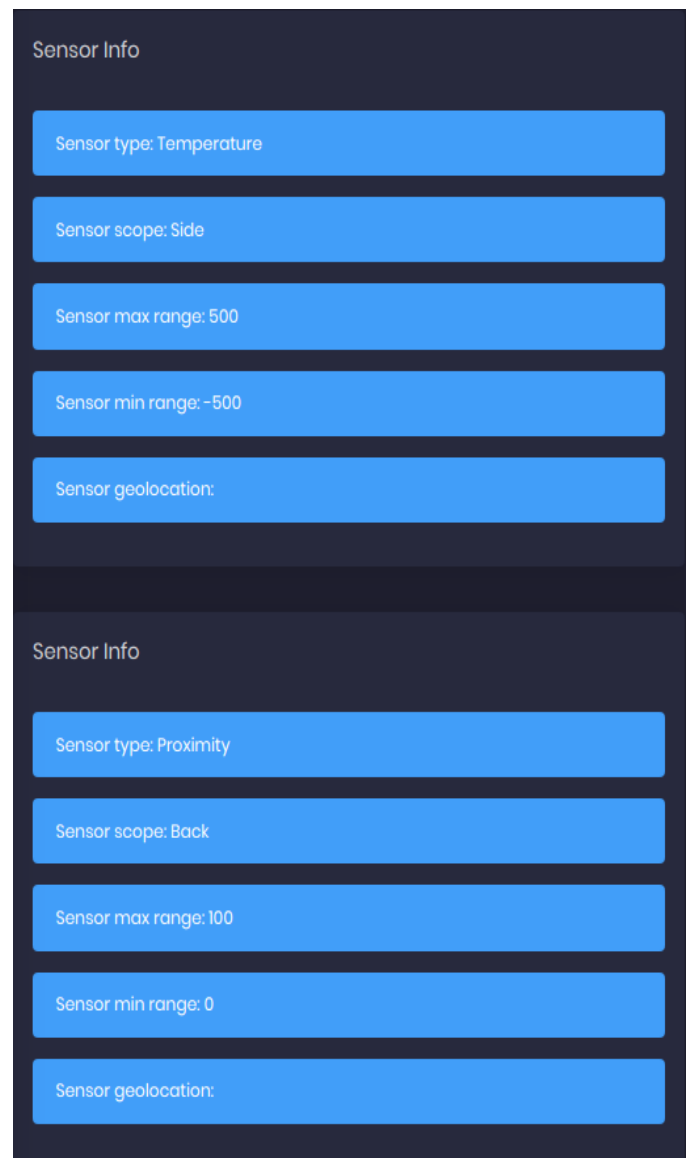


Illustration 24: Device SensorExt Rule-set

In illustration 22 and 23, we have the transaction belonging to a device. First we have the two public keys of the device, the ECC key and the ED25519 key, after that we have the rule-set representing the device registration, which contains the rules representing the new device profile. This profile contains the main data, representing the device capabilities: Vendor, CPU, GPU, RAM, and OS and if it's on acpower (Connected to the current). The profile is also composed by a list of sensors that we will see next.

Registration and consensus flowcharts for devices

Here we will explain the two approaches for consensus and its flowcharts.

Simple join request:

In this join request of a device, the devices will **not** participate in the consensus, they will be passive and they will receive updates from SENs in order to keep the session updated, in any case they will be still capable of verifying the session blockchain (see illustration 25).

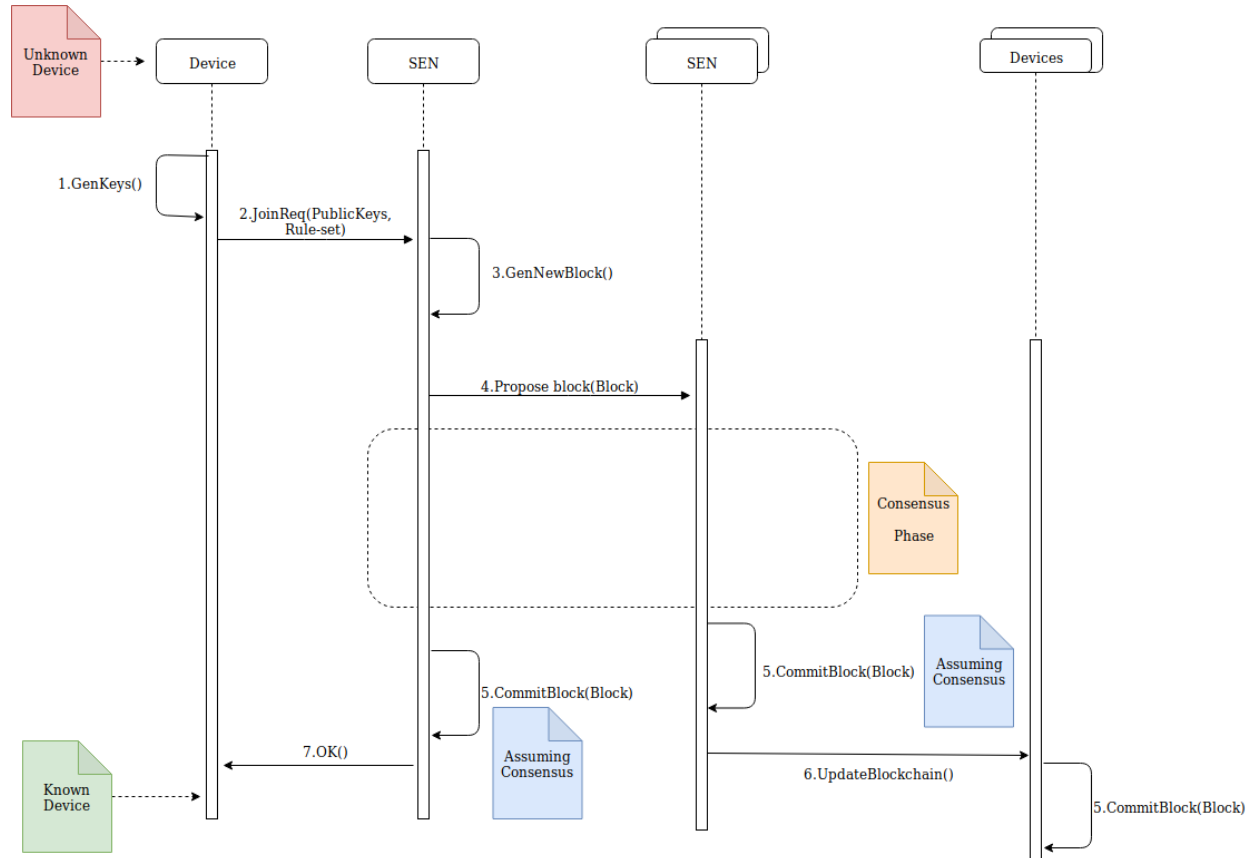


Illustration 25: Simple join request of a device using the SEN consensus protocol

In illustration 25, we have the flowchart of this request, the flowchart goes as following: (1) a new set of keys is generated in the device. (2) The candidate device sends a device registration message to the target SEN with its public keys and the desired rule-set with its profile. (3) A new block is generated with the device transaction, containing the public keys and rule-set for the device registration. (4) The newly created block is proposed to other SENs in order to start the consensus phase distributing it among all SENs. (Consensus) we won't cover the consensus flow in this diagram, as we will assume that it has gone correctly and we have consensus for the proposed block. (5) The block is committed; from this point the device is valid in the session. (6) The session is updated on the devices, as they are out of the consensus phase and do not know the newly added block. (7) An OK message is returned to the device informing that the registration has been performed correctly.

Complex join request:

In this join request of a device, the devices will participate in the consensus, they will be active and they will receive consensus messages from SENs periodically (see illustration 26).

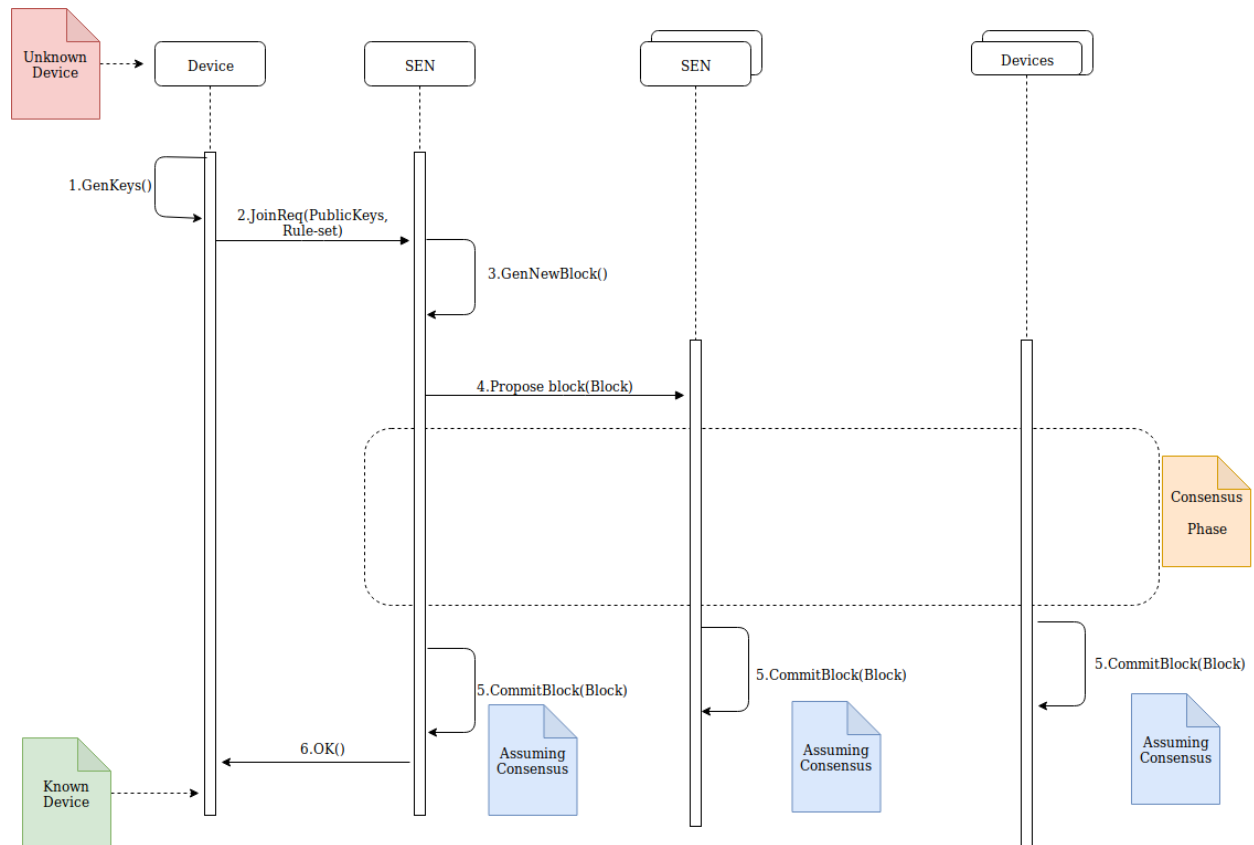


Illustration 26: Complex join request of a device using an advanced consensus algorithm

In illustration 26, we have the flowchart of this request, the flowchart goes as following: (1) a new set of keys is generated in the device. (2) The device sends a device registration message to the target SEN with its public keys and desired rule-set. (3) A new block is generated with the device transaction, containing the public keys and rule-set for the device registration. (4) The Newly created block is proposed to other SENs in order to start the consensus phase distributing it among all SENs and devices. (Consensus) We won't cover the consensus flow in this diagram, as we will assume that it has gone correctly and we have consensus for the proposed block. (5) The block is committed; from this point the device is valid in the session. (6) An OK message is returned to the device informing that the registration has been performed correctly.

4.4.5 Authentication

In order to authenticate to other devices belonging to the session a TLS connection it's used, we can simply use the public ECC key stored in the session, this connection will always be private and secure.

Establishing a TLS connection:

To illustrate the explanation, let's suppose a scenario with a device (device0) that wants to establish a secure connection to another device (device1). First of all, device0 needs to know the device1 public key. The public key is the identifier of the device1 in the session. Then the session can be used to search for a device with the desired capabilities (e.g. sensors, CPU, RAM...), or the desired public key. Once the device0 knows the device1 public key, if it doesn't know the device IP, it can query a SEN with the public key of the device1, and it will answer with the last known IP of the provided public key. Once the IP is known and reachable by the device0, it can try to establish a TLS connection with the device1. Since the device0 already knows the device1 public key, it can verify the identity through the TLS connection of such device (see illustration 27).

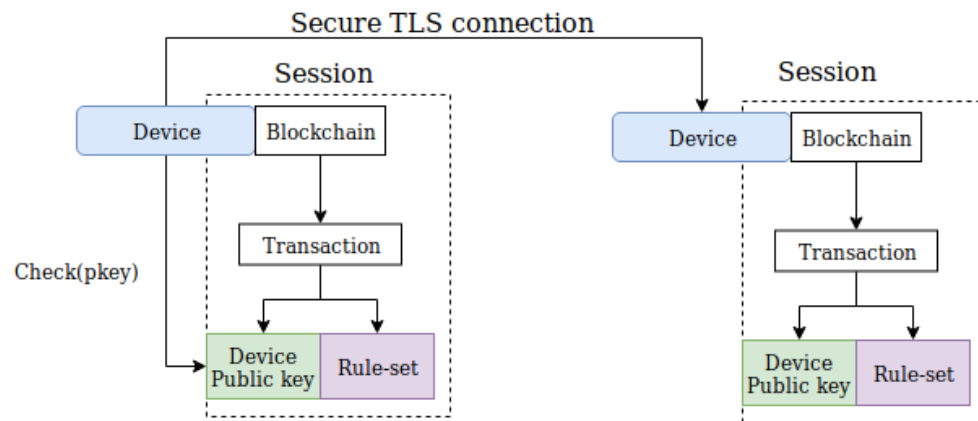


Illustration 27: Establishing a TLS connection

Receiving a TLS connection:

Another scenario could be that a device (device0) receives a TLS connection from another device (device1). The receiver can query the session (blockchain) with the device1 public key in order to check if the device1 is registered into the system. If it's registered into the system, device0 can obtain the device1 profile that is stored into the session (see illustration 28).

Then with that data it can decide what to do with the incoming connection.

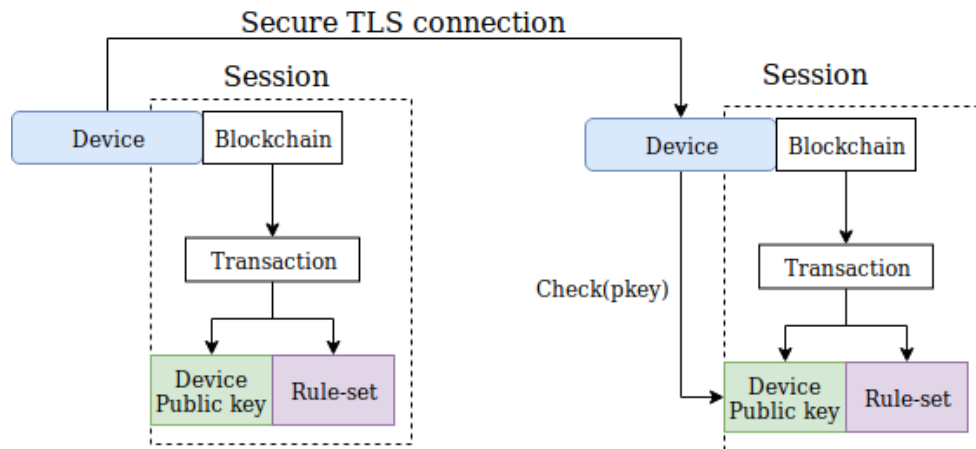


Illustration 28: Receiving a TLS connection

4.4.6 Validation

We can use the public ED25519 key in order to sign data, send it or broadcast it through the network and later on verify it in order to assure that the data is send by the expected device and it hasn't been modified.

Sending signed data:

In this case, a device (device0) wants to send signed data to any device in the session. First, it must use its private ED25519 key in order to sign the data. Then once signed, the data can be send through the network to other devices (see illustration 29).

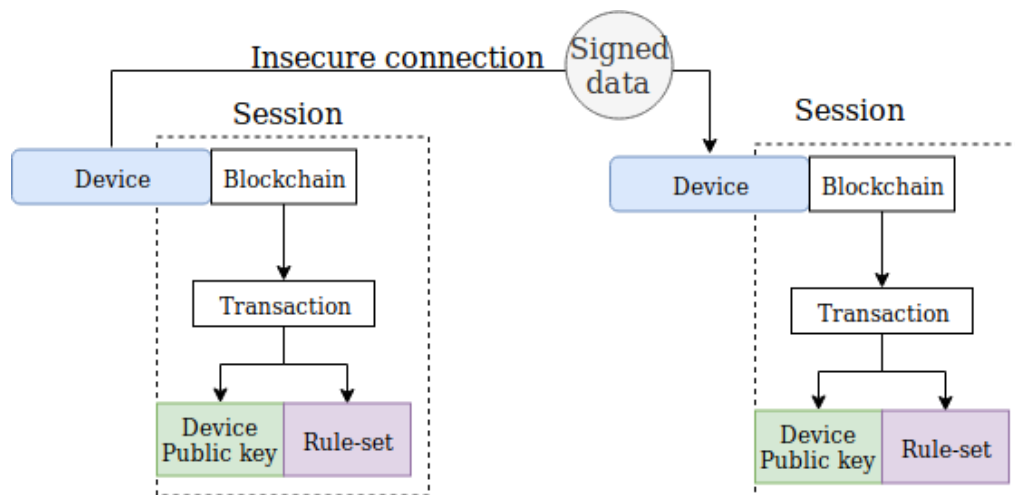


Illustration 29: Device sending signed data over insecure channel

Receiving signed data:

Continuing from the previous case, now the data sent by the device (device0) went to another device (device1), from a direct connection or a broadcast. First, it must know from which device the data is expected to come. Second, it extracts the public ED25519 key of the expected device from the session and then finally it verifies the data using the public ED25519 key (see illustration 30).

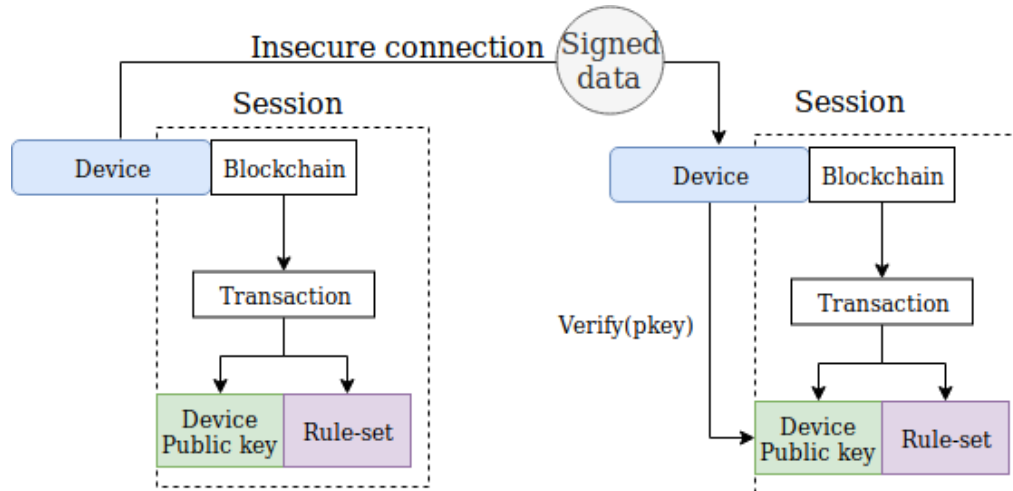


Illustration 30: Device receiving and verifying signed data over insecure channel

4.4.7 Extensions

The presented rule-set is completely extensible, and a new type of rule could be created in order to customize and fit the session into more specialized project architectures and needs.

For example, project explained in [24] could use this technology in order to secure its own custom session data distribution. However, it requires a slightly customized session. Since our session can be customized, this can be easily accomplished with our project.

4.6 Programming language used

The chosen programming language for this project was GO[ref] (Golang) because it provides one of the best repositories of security libraries out there. Go is a highly concurrent language, instead of threads it uses its go specific implementation called go routines, which save space and time over threads. Thus, it is a powerful language for web services and network related applications.

There are other more popular languages such as python, java or even C, but those are harder to work with when we have to apply security libraries, while Go keeps an active community that is developing new security libraries that can be used directly from Github and can be adapted perfectly to our needs.

In conclusion, Go was the best option for this project.

4.7 Prototype code structure

The prototype code is highly modular. All the code is structured in libraries and divided into 4 main routines.

- The routine that handles the requests from SENS.
- The routine that handles the requests from devices.
- The routine that handles the blockchain.
- The routine that manages the web interface.

All those routines communicate among them with shared channels, one per routine, when a routine wants to request an operation from another routine it sends a message to the routine channel queue and awaits for a response. Using this architecture allows the routines to be highly independent between them while avoiding any deadlock risk.

We decided to use this approach because it enables hot changes by disabling any routine and thus allowing a better debug and easier configuration on the testing and validation scenarios. All the configuration of the code parameters is written on a .env file, as environment variables. Then we can simply change the environment of the code in order to execute it with completely different configurations to perform the various tests and validations.

5 VALIDATION OF THE USE CASES

In this section we will perform a series of validations to validate the proposed architecture. First in point 5.1 the methodology used for those validations will be presented. Then on point 5.2 the main validations and its results will be shown. Finally on 5.3 the main steps to replicate the scenario will be explained.

5.1 Methodology

The methodology followed in order to tests the prototype in the following point use cases is a series of simple tests performed on a docker environment. Each test aims to validate one specific functionality of the architecture prototype. Docker allows us to deploy as many SENs and devices as required, so we can perform controlled scalability tests in a virtual network. It also allows us to check the status of the whole test from the docker run time interface, and gather the test results.

The use of docker allows us to launch almost 500 instances in our server infrastructure, thus is one of the best options to run scalability tests and one of the best options in order to build a huge testing network. While, at the same time, it allows the tests to be replicated on any docker capable machine, so the tests could be run at any environment as long as it has a correctly setup docker server and the required specs.

5.2 Validation

On this section, we will explain each use case theoretically, what we do and what we expect to will obtain. To start, we will create a SEN0 in order to initialize a new session(1). Then the use cases of adding a new SEN(2) and a new device(3) into the already existing session. Later on, we will validate the device connections by performing TLS connections between devices(5-7) and validating signed data(4-6-8). After that, we will test the search for devices and sensors in the session(9), then we will add a gateway(10) and blacklist a compromised device(11).

1. Adding SEN0 (Session set up demo)

Description	In this use case we will add the sen0 or first SEN into to the session, in order to initialize a new session. The first SEN will have a specific rule-set informing that indeed it's a sen0 and with a custom extension field that will customize the whole session behavior. For the prototype this customization is limited to the consensus algorithm used, and there is only one option for the moment. The first SEN pair of keys are from extreme relevance in the session, the public keys will be used in order to identify the whole session to other users.
Methodology	The testing methodology will be the launch of a single SEN instance in an empty network, thus the SEN will become the sen0 of a new session.
Expected result	The expected result is a new session blockchain with a single block containing the SEN0 for the new session.
Result	Passed

The used rule-set was the following:

```
ruleset := &Ruleset{
  Ruleset: &Ruleset_Sen_Scope{
    Sen_Scope: &Sen{
      AllowBlacklistDevices: true,
      AllowBlacklistSens:   true,
      AllowNewDevices:      true,
      AllowNewSens:         true,
      Extensions: &SenExtensions{
        Sen0: true,
        ConsensusAlgorithm: "debug",
      },
    },
  },
},
```

2. Adding a new SEN (Rule-set demo)

Description	In this use case we will add a new SEN into the previously created session.
Methodology	We will launch a new SEN instance in the already created SEN session, the new SEN will bootstrap into the SEN0 and perform a registration request to become a new SEN, providing its public keys and its rule-set.
Expected result	We expect the SEN0 to verify and approve the new SEN request, then we expect the session to contain the newly added SEN.
Result	Passed

The used rule-set was the following:

```
ruleset := &blockchainBtree.Ruleset{
    Ruleset: &blockchainBtree.Ruleset_Sen_Scope{
        Sen_Scope: &blockchainBtree.Sen{
            AllowNewDevices:    true,
            AllowBlacklistDevices: true,
            AllowBlacklistSens:  true,
            AllowNewSens:        true,
        },
    },
}
```


3. Adding a new Device (Rule-set demo)

Description	In this use case we will add a new device into the previously created session.
Methodology	We will launch a new device instance in the already created session, the new device will bootstrap into the SEN0 and perform a registration request to become a new registered device in the session, providing its public keys and its rule-set.
Expected result	We expect the SEN0 to verify and approve the new device request, then we expect the session to contain the newly added device.
Result	Passed

The used rule-set was the following:

```
var sensors []*blockchainBtree.SensorExt
    sensor0 := &blockchainBtree.SensorExt{
        Type:      "humidity",
        Scope:      "Front,Top,Left",
        MaxRange:   "100",
        MinRange:   "0",
    }
    sensor1 := &blockchainBtree.SensorExt{
        Type:      "Proximity",
        Scope:      "Front,Top,Right",
        MaxRange:   "100",
        MinRange:   "0",
    }
    sensor2 := &blockchainBtree.SensorExt{
        Type:      "Temperature",
        Scope:      "Side",
        MaxRange:   "500",
        MinRange:   "-500",
    }
    sensor3 := &blockchainBtree.SensorExt{
        Type:      "Proximity",
        Scope:      "Back",
        MaxRange:   "100",
        MinRange:   "0",
    }
    sensors = append(sensors, sensor0)
    sensors = append(sensors, sensor1)
    sensors = append(sensors, sensor2)
    sensors = append(sensors, sensor3)
```

```

ruleset := &blockchainBtree.Ruleset{
    Ruleset: &blockchainBtree.Ruleset_DeviceScope{
        DeviceScope: &blockchainBtree.Device{
            Data: &blockchainBtree.Data{
                Vendor: "MSI",
                CPU: "Intel",
                GPU: "NVIDIA",
                RAM: "16GB",
                OS: "Linux",
                ACpower: false,
            },
            SensorExt: sensors,
        },
    },
}

```

4. Device using the session blockchain to validate data

Description	We want to use the session in order to validate data send by a testing device from another device, we will call the sender device deviceS and the receiver device deviceR.
Methodology	We will set up the deviceS in order to send signed data, that data will be signed by its ED private key. The data will be send to the deviceR, the deviceR will then use the session in order to know the deviceS public key and verify the data.
Expected result	The expected result is the sending of the signed data by the deviceS and then the validation of the data by the receiver deviceR.
Result	Passed

5. Device using the session blockchain to establish TLS

Description	We want to use the session in order to establish a secure and authenticated TLS connection between two devices. We will call the starter device deviceS and the receiver device deviceR.
Methodology	We will set up the deviceS in order to start a secure TLS connection with deviceR. To do so, we will search the session for the deviceR in order to get its ECC keys, then we will be able to establish a secure TLS connection with the deviceR.
Expected result	The expected result is the establishment of a TLS connection from deviceS to deviceR, using the keys obtained from the session.
Result	Passed

6. Anonymous device using blockchain to validate data

Description	We want to use the session in order to validate data send by a testing device from another unregistered device, we will call the sender device deviceS and the receiver device deviceR.
Methodology	We will set up the deviceS in order to send signed data, that data will be signed by its ED private key. The data will be send to the deviceR, the deviceR will then use the session in order to know the deviceS public key and verify the data.
Expected result	The expected result is the sending of the signed data by the deviceS and then the validation of the data by the receiver deviceR. But deviceR will never be registered in the session, it will just have a copy of it in order to know its participants.
Result	Passed

8. Simple device broadcasting data

Description	We want to broadcast sensor data from a device, the data will be signed by its ED key
Methodology	We will fake a sensor inside a device in order to broadcast some data to the network, the data will be signed by its private ED key.
Expected result	We expect the data to be correctly signed and broadcasted.
Result	Passed

9. Searching the session for a specific sensor

Description	We want to be able to find specific sensors in the session, for example a device will need to search for specific sensors in order to execute services of the Fog.
Methodology	We will do a query internally on the device, we don't expect to open any API in order to get the information from the outside, but we expect to be able to query the session for specific sensors.
Expected result	We expect to obtain a sensor and the device that owns the sensors, matching our query if it exists in the session.
Result	Passed

10. Gateway (Rule-set demo)

Description	We want to register a gateway, the gateway will have two devices with its pertinent sensors, and the keys for the gateway devices will be emulated, as we don't have such physical or emulated structure for the test.
Methodology	We will set up a gateway, register it with tree devices and its sensors.
Expected result	We expect the gateway to be added into the session correctly.
Result	Passed

The used rule-set was the following:

```
var sensors []*blockchainBtree.SensorExt
    sensor0 := &blockchainBtree.SensorExt{
        Type:      "humidity",
        Scope:     "Front,Top,Left",
        MaxRange:  "100",
        MinRange:  "0",
    }
    sensor1 := &blockchainBtree.SensorExt{
        Type:      "Proximity",
        Scope:     "Front,Top,Right",
        MaxRange:  "100",
        MinRange:  "0",
    }
    sensor2 := &blockchainBtree.SensorExt{
        Type:      "Temperature",
        Scope:     "Side",
        MaxRange:  "500",
        MinRange:  "-500",
    }
    sensor3 := &blockchainBtree.SensorExt{
        Type:      "Proximity",
        Scope:     "Back",
        MaxRange:  "100",
        MinRange:  "0",
    }
    sensors = append(sensors, sensor0)
    sensors = append(sensors, sensor1)
    sensors = append(sensors, sensor2)
    sensors = append(sensors, sensor3)
    // define rule-set for devices
    complexDev1 := &blockchainBtree.ComplexDevice{
        Keys: nil,
        Data: &blockchainBtree.Data{
            Vendor: "Raspberry",
            CPU:    "ARM",
            GPU:    "ARM",
            RAM:    "2GB",
            OS:    "Linux",
            ACpower: false,
        },
        SensorExt: sensors,
    }
    complexDev2 := &blockchainBtree.ComplexDevice{
        Keys: nil,
        Data: &blockchainBtree.Data{
            Vendor: "Raspberry1",
            CPU:    "ARM",
            GPU:    "ARM",
            RAM:    "2GB",
            OS:    "Linux",
            ACpower: false,
        },
        SensorExt: sensors,
    }
var complexDevices []*blockchainBtree.ComplexDevice
```

```

complexDevices = append(complexDevices, complexDev1)
complexDevices = append(complexDevices, complexDev2)
// Define rule-set for gateway
policies := &blockchainBtree.Policies{
    Scope: &blockchainBtree.Policies_GatewayDeviceScope{
        GatewayDeviceScope: &blockchainBtree.GatewayDevice{
            Data: &blockchainBtree.Data{
                Vendor: "Raspberry",
                CPU: "ARM",
                GPU: "ARM",
                RAM: "2GB",
                OS: "Linux",
                ACpower: false,
            },
            Devices: complexDevices,
        },
    },
}

```

11. Blacklist (Rule-set demo)

Description	We want to blacklist a device, by providing the device public key.
Methodology	We will issue a device blacklist from a SEN.
Expected result	We expect the device to become blacklisted in the session.
Result	Passed

The used rule-set was the following:

```

ruleset := &blockchainBtree.Ruleset{
    Ruleset: &blockchainBtree.Policies_Blacklist{
        Blacklist: &blockchainBtree.Blacklist{
            FromTimestamp: "Blacklist testing",
        },
    },
}
node := networkMap.GetNode([]byte("45 45 45 45 45 66 69 71 73 78 32 80 85 66
76 73 67 32 75 69 89 45 45 45 45 10 77 70 107 119 69 119 89 72 75 111 90
73 122 106 48 67 65 81 89 73 75 111 90 73 122 106 48 68 65 81 99 68 81 103 65
69 82 65 43 87 81 97 111 87 107 65 108 48 77 105 75 85 118 72 78 52 103 49
107 122 65 89 97 105 10 90 99 103 90 55 109 90 89 53 81 55 97 75 109 88 81 89
87 80 75 85 115 114 102 104 105 115 49 82 78 89 65 65 53 77 79 66 120 75 48
85 79 101 77 73 70 122 49 48 74 100 82 68 111 121 107 55 81 61 61 10 45 45 45
45 45 69 78 68 32 80 85 66 76 73 67 32 75 69 89 45 45 45 45 10"))

```

5.3 Testing scenario replication

In this chapter, we will explain the set of tools and scripts in order to replicate the prototype testing scenario in any capable environment, step by step:

1. The first requirement would be docker, docker is required in order to run the containers of the SENs and the devices.
2. The second requirement would be an internal docker network, in order to connect the containers.
3. The third requirement would be the build of the provided docker files, in order to build the base images for the SENs and the devices, each SEN must be built in a separate image, as we require to previously know its keys, devices can all be built from the same images as the keys are auto generated.
4. Once this steps are met, and we have a functional docker machine with a configured docker virtual network and the build images, we are ready to launch the prototype.
5. First of all we must launch the sen0 image with a provided IP, once the sen0 is up and running we can add more SENs providing its IPs and using a prebuild SEN image, (we could use one SEN image to launch them automatically as we do with the devices, but then we will have less control).
6. Once we have a minimum of one SEN we can start adding devices, a script is provided to control this phase, to add devices, stop, and remove them. The script launches n new devices and provides them with a new IP in a selected range.

We recommend the use of Portainer or another docker management web interface in order to have a better view and control of the whole prototype testing scenario.

6 FUTURE WORK

In this chapter we will present the points that are related to this project but are still open.

- Perform a real testing in a pilot, in order to test the real network applications during a period of time, and monitor the session behavior.
- The consensus algorithm should be swapped by RBFT or Indy RBFT, in order to be tested on huge environments.
- In order to save space and tolerate failures we would require to perform the whole peer lookup with a DHT, Kademlia will be the best choice here.
- In order to allow third parties to use this technology it will be a plus to have an API, then this project could be a black box that would be used to secure a node, the node will interact with this black box but it will never know what happens inside, it will just be provided with security out of the box.

7 CONCLUSIONS

In this project we have proposed and developed a new secure architecture for the Fog using blockchain technologies.

The main idea here is a distributed security architecture, where the fog session is stored in the blockchain, and it is shared among all the fog devices. Then, it is used to authenticate and verify the devices. Nevertheless, this architecture requires the involvement of all the Fog devices. At the end, the devices are the ones accountable for the security and this project provides them with the tools to be secure (the session).

The main advantage of this architecture compared with other security centralized approaches is the scalability and immutability of this model. This model has all the blockchain characteristics. First it's completely distributed. Second thanks to the immutability property, while at least one device and one validator are up, the session will be preserved.

In this proposal, the network is formed by Fog devices, and the idea is to have proposed different type of devices depending on its role: devices (normal, simple, anonymous), SENs (validators) and Gateways. While the devices and gateways are mere actuators and members of the session, the SENs have the role of validating and providing security to the whole session blockchain.

Apart of defining the role of the different devices, the main structure of the blockchain, the block, has been defined, including all the needed fields. The novelty of this block compared with other blockchains is the link between the transaction keys and the rule-set. This relation defines the session, and the actions that happen on it.

Accomplished objectives from the objectives overview:

- ✓ Proposed new blockchain model to secure a Fog area.
- ✓ Delivered the architecture plus the prototype code for the proposed model.
- ✓ Used the prototype to test the model validity with a series of testing and validation scenarios.
- ✓ The model successfully secures a Fog area, and provides Fog profiles to the Fog devices.
- ✓ The blockchain is used as the Fog session holder.
- ✓ All the devices are able to register into the session.
- ✓ All the devices are able to use the session to establish secure connections.

The blockchain technology is a good tool for this objective, but it has to be used carefully as it can drawback and overload the Fog with useless and irrelevant residual data for the devices. The Fog has a constrained amount of memory on its devices, as it can't scale as the cloud does, and the blockchain for definition will always grow.

In order to publish this prototype and made it usable for others on production Fog environments, at minimum the consensus algorithm and DHT stated in the future work point 6 must be added. Although the prototype can be used as it is, on other prototype projects or projects where the testing scenarios are relatively small, as the performance won't be affected by the lack the future work points.

8 ACKNOWLEDGMENTS

I would like to thank my thesis supervisor Eva Marin Tordera for its expert guidance and my thesis tutor Beatriz Otero Calviño for its expert comments and suggestions. I also would like also to thank the whole CRAAX research group for its constant support, Jordi Garcia Alminyana, Sergi Sanched López and specially its director Xavier Masip Bruin. Finally, I would like to thank my friend Alejandro Jurnet Bolarin for the countless discussion hours about this thesis and other related topics that he had to endure.

Apart from this I would like also to credit the CRAAX research group for letting me use the necessary infrastructure and equipment in order to test this thesis.

9 REFERENCES

- [1] A White Paper from the Sovrin Foundation “Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust” version 1.0 January 2018
- [2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. ACM, New York, NY, USA, Article 30, 15 pages. DOI: <https://doi.org/10.1145/3190508.3190538>
- [3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12)*. ACM, New York, NY, USA, 13-16. DOI=<http://dx.doi.org/10.1145/2342509.2342513>
- [4] Vitalik Buterin “Ethereum White Paper A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM”
- [5] Satoshi Nakamoto “Bitcoin: A Peer-to-Peer Electronic Cash System”
- [6] Gramoli, Vincent. “On the Danger of Private Blockchains (When PoW can be Harmful to Applications with Termination Requirements).” (2016).
- [7] Z. Zheng, S. Xie, H. Dai, H. Wang, "An overview of blockchain technology: Architecture consensus and future trends", *Proc. IEEE Int. Congr. Big Data Big Data Congr.*, pp. 557-564, Jun. 2017.
- [8] Hyperledger Indy “<https://www.hyperledger.org/projects/hyperledger-indy>” date: april 29 2019
- [9] P.-L. Aublin, S. B. Mokhtar, V. Quema, "Rbft: Redundant byzantine fault tolerance", *ICDCS*, 2013.
- [10] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *Proceedings of the third symposium on Operating systems design and implementation (OSDI '99)*. USENIX Association, Berkeley, CA, USA, 173-186.
- [11] Torre Dominique, Sothearath Seang “Proof of Work and Proof of Stake consensus protocols: a blockchain application for local complementary currencies” International Symposium on Money, Banking and Finance, sciences Po Aix, 2018
- [12] Kwon, Jae Kyun. “Tendermint : Consensus without Mining.” (2014).
- [13] Peercoin “<https://docs.peercoin.net/>” date: april 25 2019
- [14] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2010. Zyzzyva: Speculative Byzantine fault tolerance. *ACM Trans. Comput. Syst.* 27, 4, Article 7 (January 2010), 39 pages. DOI=<http://dx.doi.org/10.1145/1658357.1658358>

- [15] Weise, Joel. "Public key infrastructure overview." *Sun BluePrints OnLine, August* (2001): 1-27.
- [16] Steiner, Jennifer G., B. Clifford Neuman, and Jeffrey I. Schiller. "Kerberos: An Authentication Service for Open Network Systems." *Usenix Winter*. 1988.
- [17] Hammi, Mohamed Tahar & Bellot, Patrick & Serhrouchni, Ahmed. (2018). BCTrust: A decentralized authentication blockchain-based mechanism. 1-6. 10.1109/WCNC.2018.8376948.
- [18] An Introduction to Hyperledger. Available online at "https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf" date: april 29 2019
- [19] Hyperledger Fabric "<https://www.hyperledger.org/projects/fabric>" date: april 29 2019
- [20] Lopez, Julio, and Ricardo Dahab. "An overview of elliptic curve cryptography." (2000).
- [21] Wu, Di, et al. "Understanding peer exchange in bittorrent systems." *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2010.
- [22] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron (Eds.). Springer-Verlag, London, UK, UK, 53-65.
- [23] Protocol Buffers "<https://developers.google.com/protocol-buffers/>" date: may 03 2019
- [24] Alejandro Jurnet Bolarin, "Control resilience in a F2C scenario" Universitat Politècnica de Catalunya, to be published on July 2019.

10 ANNEX

Codebase for the prototype can be found in the following Gitlab URL. The cosebase is private and any access must be requested to this thesis tutor or supervisor.

<https://gitlab.com/PauMarcera/securedgenodearchitecture>