

## Replacing Public Key Infrastructures (PKI) by blockchain IoT devices security management

**Auteur :** Champagne, Loïc

**Promoteur(s) :** Leduc, Guy; 12788

**Faculté :** Faculté des Sciences appliquées

**Diplôme :** Master en sciences informatiques, à finalité spécialisée en "computer systems security"

**Année académique :** 2020-2021

**URI/URL :** <http://hdl.handle.net/2268.2/11608>

---

### Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

---

# Replacing Public Key Infrastructures by blockchain IoT devices security management

**Author:**

Champagne Loïc

**Supervisors:**

Prof. G. Leduc  
E. Tychon (Cisco)



Thesis submitted for the degree of  
Master in Computer Science with a professional  
focus on "Computer systems security"  
120 credits

Faculty of Applied Science

UNIVERSITY OF LIÈGE

Academic year 2020-2021



# **Replacing Public Key Infrastructures by blockchain IoT devices security management**

Champagne Loïc

© 2021 Champagne Loïc

Replacing Public Key Infrastructures by blockchain IoT devices security  
management

<https://matheo.uliege.be/>

# Abstract

Champagne Loïc: "Replacing Public Key Infrastructures by blockchain IoT devices security management."

Master of Science Thesis

University of Liège

June 2021

---

On the Internet, Public Key Infrastructure (PKI) is the most advanced credential management system. However, the standard PKI relies on certificate authorities (CAs) which have delivered certificates to the wrong people in the past for questionable reasons. Indeed, these CAs represent a corruptible central point that this work aims to remove. This was done by adapting the PKI to a decentralized framework based on blockchain smart contract. This solution is essentially targeted toward the Internet of Things (IoT) that currently lacks a scalable system for managing keys and identities (i.e., a standard PKI framework). Unlike CA-based PKIs, our framework delivers auditability natively which provides a proof of the framework integrity. In order to adequately test our solution, we designed and implemented a proof of concept. The smart contract was written in solidity and is deployed on the Kovan test net. After testing our solution on an Ubuntu core virtual machine, we found that the solution has a very small footprint and is therefore adapted to the IoT ecosystem.

Keywords: IoT, PKI, Decentralized, Smart contract, Blockchain, Authentication, Security.

# Preface

First and foremost, I would like to express my deepest gratitude to my supervisors, Prof. Guy Leduc and Emmanuel Tychon, for their precious advice and for their attentive supervision of this thesis. Thank you for giving me the opportunity to work on such an interesting topic.

I would also like to kindly thank my friends and family for their support throughout the year, and especially to my friend Hugo Fooy for his proof-reading of this thesis, acute critics, and unconditional friendship.

Liège, 27th May 2021

Loïc Champagne

# Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>Acronyms</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 C-PKI drawbacks for IoT . . . . .	1
1.2 Defining goals . . . . .	3
1.3 Research questions . . . . .	3
<b>I Background</b>	<b>5</b>
<b>2 Theoretical background</b>	<b>6</b>
2.1 Public Key Infrastructure . . . . .	6
2.1.1 CA based PKI (C-PKI) . . . . .	7
2.2 Blockchain . . . . .	9
2.2.1 Proof-of-Work and Consensus . . . . .	9
2.2.2 Smart contract . . . . .	10
2.2.3 Name-Value Storage . . . . .	10
<b>3 State of the Art</b>	<b>11</b>
3.1 Blockchain based solutions . . . . .	11
3.1.1 Certcoin and IoT-PKI . . . . .	11
3.1.2 Scpki . . . . .	12
3.2 CA based solutions . . . . .	12
3.2.1 EST over secure CoAP . . . . .	12
3.2.2 PKI4IoT . . . . .	13
<b>II The Framework</b>	<b>14</b>
<b>4 Security requirements and threat model</b>	<b>15</b>
<b>5 Architecture</b>	<b>17</b>
5.1 IoT Controller . . . . .	17
5.1.1 Nature of the controller . . . . .	17
5.1.2 Ethereum nodes . . . . .	18



5.2	IoT device . . . . .	19
5.2.1	Device life cycle . . . . .	19
5.2.2	Lasting certificate for the IoT device . . . . .	20
5.3	Device enrollment . . . . .	21
5.4	Device certificate update . . . . .	22
5.5	Device certificate revocation . . . . .	24
5.6	Device lookup . . . . .	25
5.7	Smart contract . . . . .	25
5.7.1	Smart contract vs. NVS . . . . .	25
5.7.2	Data stored on the blockchain . . . . .	25
5.7.3	Setter/Getter functions . . . . .	26
5.8	Overall view of the architecture . . . . .	27
5.9	Possible improvements . . . . .	28
<b>III</b>	<b>Framework analysis</b>	<b>29</b>
<b>6</b>	<b>Scalability of the framework</b>	<b>30</b>
6.1	The DCS trilemma . . . . .	30
6.2	Implication of the DCS trilemma on the framework . . . . .	30
6.3	Solutions to that limitation . . . . .	32
<b>7</b>	<b>Security assessment of the developed framework</b>	<b>33</b>
7.1	Study of the feasibility and security of the enrollment stage:	33
7.2	Study of the feasibility and security of the update and revocation stages: . . . . .	34
7.3	Security limitation . . . . .	35
<b>IV</b>	<b>The Proof of Concept</b>	<b>36</b>
<b>8</b>	<b>Technologies</b>	<b>37</b>
8.1	Why solidity and public Ethereum . . . . .	37
8.2	Ethereum Virtual Machine . . . . .	38
<b>9</b>	<b>Implementation</b>	<b>39</b>
9.1	Device/controller interactions . . . . .	39
9.1.1	Device/Controller discovery . . . . .	39
9.1.2	Device/Controller communications . . . . .	40
9.1.3	Construction of the custom X.509 certificates . . . . .	40
9.2	Smart contract . . . . .	41
9.2.1	Structure of the contract . . . . .	41
9.2.2	Deployment . . . . .	41
9.3	Improvements . . . . .	42
<b>10</b>	<b>Results</b>	<b>44</b>
10.1	Experimental setup . . . . .	44
10.2	Device benchmarks . . . . .	44
10.2.1	Memory usage . . . . .	45

10.2.2	Network usage . . . . .	46
10.2.3	CPU usage . . . . .	46
10.2.4	Disk usage . . . . .	48
<b>V</b>	<b>Conclusion</b>	<b>49</b>
<b>11</b>	<b>Discussion</b>	<b>50</b>
11.1	Comparison of the different solutions . . . . .	50
11.2	Research questions . . . . .	51
11.3	Future works . . . . .	51
<b>12</b>	<b>Final conclusion</b>	<b>52</b>
	<b>Appendices</b>	<b>57</b>
<b>A</b>	<b>Ubuntu Core key characteristics</b>	<b>58</b>
<b>B</b>	<b>Additional system benchmarks of the device</b>	<b>59</b>

# List of Figures

1.1	Ranking of IoT Security Weaknesses. Source: IoT Analytics .	2
2.1	Steps to verify the integrity of a message using digital signature. Source: [46] . . . . .	7
2.2	Direct trust vs. third party trust between individuals. . . . .	8
2.3	Flow of a certificate request in a C-PKI . . . . .	8
3.1	The certificate enrollment process using EST. . . . .	12
4.1	Classification of threat models for Blockchain. Source [15] .	16
5.1	Chart of the storage needed to run a full node. Source: Etherscan.io . . . . .	18
5.2	Life cycle of the IoT device. . . . .	20
5.3	Device enrollment into the PKI (explanation can be found in the Section 5.3 : Device enrollment ) . . . . .	21
5.4	Device key update (explanation can be found in the Section 5.4) . . . . .	23
5.5	Suggested cryptoperiods for key type. Source: section 5.3.6 in the report [3] . . . . .	24
5.6	Structure of the data stored on the smart contract hash map	26
5.7	Overview of the whole architecture . . . . .	27
6.1	Pie chart showing the top 25 miners of the Ethereum Blockchain. Source: Etherscan.io . . . . .	31
6.2	Illustration of the DCS trilemma. . . . .	31
10.1	Amount of used memory in kilobytes. (This does not take into account memory used by the kernel itself.) . . . . .	45
10.2	Amount of memory in kilobytes needed for current workload. (This is an estimate of how much RAM/swap is needed to guarantee that there never is out of memory.) . . .	45
10.3	Amount of Kb sent and received over the network. . . . .	46
10.4	Percentage of CPU usage by user processes. . . . .	46
10.5	Number of system page fault. . . . .	47
10.6	Number of interrupts. . . . .	47

# List of Tables

11.1	Comparison of our framework with PKI4IOT . . . . .	50
A.1	Ubuntu Core key characteristics. Source: [13] . . . . .	58

# Acronyms

- API** Application Programming Interface. 19, 43
- CA** Certificate Authority. 1, 2, 4, 7, 11, 13, 50–52
- COAP** Constrained Application Protocol. 12, 13, 28, 40, 51
- CPU** Central processing unit. 44, 46
- CSR** certificate signing request. 7, 13
- DCS** Decentralized Consensus Scale. 30
- DNS** Domain Name System. 27, 28, 35, 39, 40, 42
- DoS** Denial of Service. 41
- DTLS** Datagram Transport Layer Security. 33, 34
- ECDHE** Elliptic Curve Diffie-Hellman Ephemeral. 34
- EST** Enrollment over Secure Transport. 12, 13
- EVM** Ethereum Virtual Machine. 38
- HTTP** Hypertext Transfer Protocol. 12, 13, 25, 28, 40, 51
- IETF** Internet Engineering Task Force. 40
- IoT** Internet of Things. vi, 1–3, 11, 17, 19, 20, 22, 25, 26, 33, 34, 39, 44, 50–52
- IP** Internet Protocol. 27, 35, 39, 40, 42
- NIST** National Institute of Standards and Technology. 22
- NVS** Name-Value Storage. 10, 11, 25, 51
- OS** Operating System. 45
- PKI** Public Key Infrastructure. vi, 1, 3, 4, 6–8, 11, 12, 22, 24, 25, 43, 50–52
- PoC** Proof of Concept. 39, 44, 45

**PoS** Proof of Stake. 32

**PoW** Proof of Work. 9, 10, 32

**RAM** random access memory. 44, 46

**RPC** Remote Procedure Call. 19

**RSA** Rivest–Shamir–Adleman. 42, 43

**SOTA** State of the Art. 50

**SSDP** Simple Service Discovery Protocol. 39

**SSH** Secure Shell. 44, 45

**SSL** Secure Sockets Layer. 34

**TLS** Transport Layer Security. 12, 22, 33, 34, 40

**UDP** User Datagram Protocol. 28, 33, 40

**UUID** Universally Unique Identifier. 19, 25–28, 35, 41

**VM** Virtual Machine. 44

# Chapter 1

## Introduction

The rise of low-cost hardware in conjunction with the strong desire to simplify our daily tasks tends to put forward new devices in our daily lives. Those devices are called Internet-of-Things (IoT) devices. They come in a variety of formats going from connected appliances to wearable health monitors. As a result, IoT is making our world more efficient and intelligent. Indeed, the IoT ecosystem is one of the fastest growing. One projection has more than 100 billion connected devices in use by 2050[22].

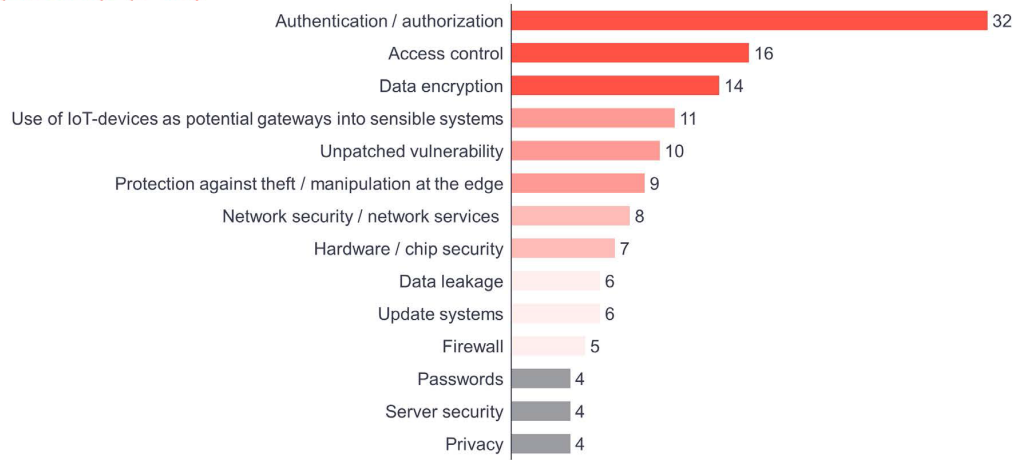
This new family of devices also comes with some downsides. The proliferation of IoT devices implies that those devices may quickly become a breeding ground for cyber attacks. As a matter of fact, those devices have far less computational power than modern computers which makes them far more difficult to secure, creating new needs for dedicated protocols and specialized personnel. Furthermore, these networked devices have access to data that can be intensely private, e.g., when you sleep, what your door lock pin code is, what you watch on TV or other media, and who and when others are in the house. Moreover, the state of the devices themselves represents potentially sensitive information. The combination of all these factors is what really pushes forward the research for new security protocols targeted at IoT devices.

This thesis will focus on the authentication side of security for IoT devices. Recently in 2017, an industry report by IoT Analytics covering the period 2017-2022 gathered input from technologists about the greatest need for improvement in IoT security, where respondents ranked authentication/authorization and access control on top of the list [23] (see Figure 1.1). The current Public Key Infrastructure (PKI) relying on Certificate Authorities (CAs), referred to as C-PKI in what follows, provides such critical security functions when IoT devices have CA-signed certificates. X.509 certificates for IoT device authentication have attracted industry interest[31] [2].

### 1.1 C-PKI drawbacks for IoT

Despite being usable, the certificates issued by CAs are not well suited for IoT devices due to the following reasons. First of all, it is difficult for the

**Question:** Where do you see the greatest need for improvement in IoT security (select 3)? (n=39)



Source: IoT Analytics

Figure 1.1: Ranking of IoT Security Weaknesses. Source: IoT Analytics

owner to manage the certificate of its IoT devices as there is no standard protocol to register, update, or revoke the certificates on those devices. Secondly, due to limitations and heterogeneity of the devices' resources, existing security solutions are not fully adapted to such an ecosystem. Besides, the combination of multiple security technologies and solutions is often needed, resulting in high costs. Furthermore if a CA is compromised, all entities relying on this CA must be updated in order to remove this trusted CA. This is a very lengthy and laborious process that may lead to big problems. Finally, not only for IoT devices, the CA ecosystem is fragile and prone to compromises and operational errors. Several times in the past, major issues have happened with this architecture all over the world [9] [39] [32]. Unfortunately, this design has demonstrated both serious usability and security shortcomings[11]. On top of that, some CAs have previously delivered certificates to the wrong individuals, or for questionable purposes. These trusted third parties serve as corruptible central points of vulnerability, each capable of jeopardizing the Internet's credibility and security. This last piece really demonstrates the huge flaw of relying on a third party to enable trust.

All these major deficiencies are also accompanied by other problems that make the use of such an infrastructure unusable on a large scale in the context of the Internet of Things. Those problems are:

- **High certificate signing cost.** Nowadays certificate price is impracticable for such a large number of IoT devices. Depending on the CA and the form of credential required, a single certificate will cost anything from \$100 to \$1,000[24]. In addition, to attract potential customers, the price of such devices must be very low. This is not possible with such expensive certificates.



- **Slow certificate signing process.** Depending on the CA and the form of certificate required, the standard CA certificate signing process will take several days<sup>1</sup>.
- **Difficulties in maintaining root certificate lists.** IoT devices have very small storage and therefore a device may not have enough storage to store the certificate lists. On top of that, these lists are often modified and thus the devices will need to be updated each time these lists are changed.

All those problems are what motivates the exploration of new alternatives to the C-PKI infrastructure. In this thesis, we propose a decentralized alternative that aims at improving the current PKI for IoT devices. On top of that, a blockchain based PKI is a path toward a community driven trust store that removes the need of a trusted third party.

## 1.2 Defining goals

The purpose of this thesis is to assess the feasibility of a decentralized alternative to the third party dependent Public Key Infrastructure (i.e., C-PKI). The two approaches are to be compared by taking into account the pros and cons of each framework. This comparison should lead to a conclusion that will state which approach is better suited for computationally constrained devices (i.e., IoT devices). Furthermore, another important aspect of this work is to test the resiliency of such an infrastructure at scale as the IoT ecosystem is envisioned to explode in the next decades.

In more detail, a decentralized IoT device identity management framework is sought as the primary goal of this work by achieving the following objectives:

- Define a decentralized IoT device management framework for identity, authentication, authorization, and accounting management based on smart contracts and distributed ledger;
- Design an authentication and authorization framework that is more user and device-centric with lower computation requirements, and that enables peer-to-peer IoT applications;
- Provide a framework that enables identity retention with features of portability and traceability throughout the device life cycle.

## 1.3 Research questions

The underlying research questions that drive our efforts are:

---

<sup>1</sup>Source: <https://www.tbs-certificates.co.uk/FAQ/en/120.html>

*RQ T.1: “Is a blockchain-based PKI a viable alternative to the widely spread CA based PKI in the context of the Internet of Things?”*

*RQ T.2: “What can a blockchain-based PKI offer more than a traditional PKI?”*

## **Part I**

# **Background**

## Chapter 2

# Theoretical background

### 2.1 Public Key Infrastructure

In a small definition, a PKI is:

*"A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption." [8]*

In order to clarify this definition, we need to define several key concepts:

- **Asymmetric encryption** is an encryption scheme where an individual possesses a pair of keys linked between each other mathematically. The first key of the pair is the public key which can be shared with anybody and the other is the private key that must remain secret. The basis of asymmetric cryptography is that it is mathematically easy to create such a key pair but very hard to guess the private key just by having the public key. In mathematics, this is known as the prime factorization problem. In this scheme, one of the two keys is used to encrypt the data and only the other key of the key pair can decrypt this data.
- **Hash functions** are one way functions that take data as input and produce a fixed size digest. Those functions are called one way because it is easy to compute the digest from the data but very hard the other way around.
- **Digital signature** is a proof of the integrity of the message as well as a proof of the message origin. This is done by creating a digest of the message and encrypting the digest with the private key of the sender. Therefore, the receiving end can decrypt the digital signature with the public key of the sender and compare the digest of the message it received with the decrypted signature. The Figure 2.1 shows this process.

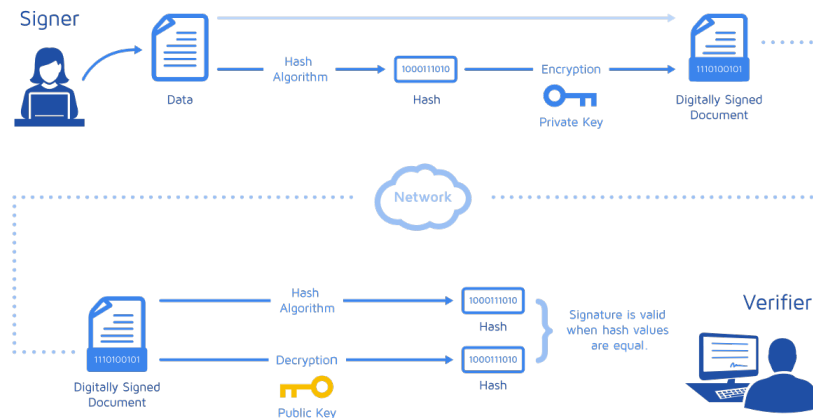


Figure 2.1: Steps to verify the integrity of a message using digital signature.  
Source: [46]

- **Digital certificate** is a digital document that proves the binding between a public key and a given entity. This proof is the issuer's (i.e., Certificate Authority) digital signature which is present on the document. In a decentralized PKI, the proof is a little bit different and the confirmation should be asked to the decentralized system.

Now that the background is laid out, we can summarize by saying that a PKI is the infrastructure that confirms the integrity of digital certificates and deals with them.

### 2.1.1 CA based PKI (C-PKI)

There are different types of PKI, the most common being the CA based PKI. This PKI uses the concept of Certificate Authority to enable trust between two parties. This concept was first introduced by Kohnfelder in 1978[26]. Despite being old, this concept is still used nowadays by private PKI companies like Digicert, Verisign and many others.

A CA is a private trusted third party that signs certificates after verifying the identity of the entity. Therefore as trust between two entities is enabled by a third party (i.e., the CA), the two entities only have a third-party trust between each other rather than a direct trust. The difference between direct trust and third party is shown on Figure 2.2.

The procedure for obtaining a signed certificate from a certificate authority is as follows (flow can be seen on Figure 2.3):

1. The client generates a public/private key pair and sends a certificate signing request (CSR) to a trusted certificate authority. This CSR contains the client's public key and information. All this information will be shown on the corresponding certificate if it is authorized;
2. The CA checks to see if the details on the CSR is true. If that is the case, it creates and signs a certificate with its (the CA's) private key before handing it (the certificate) over to the customer.

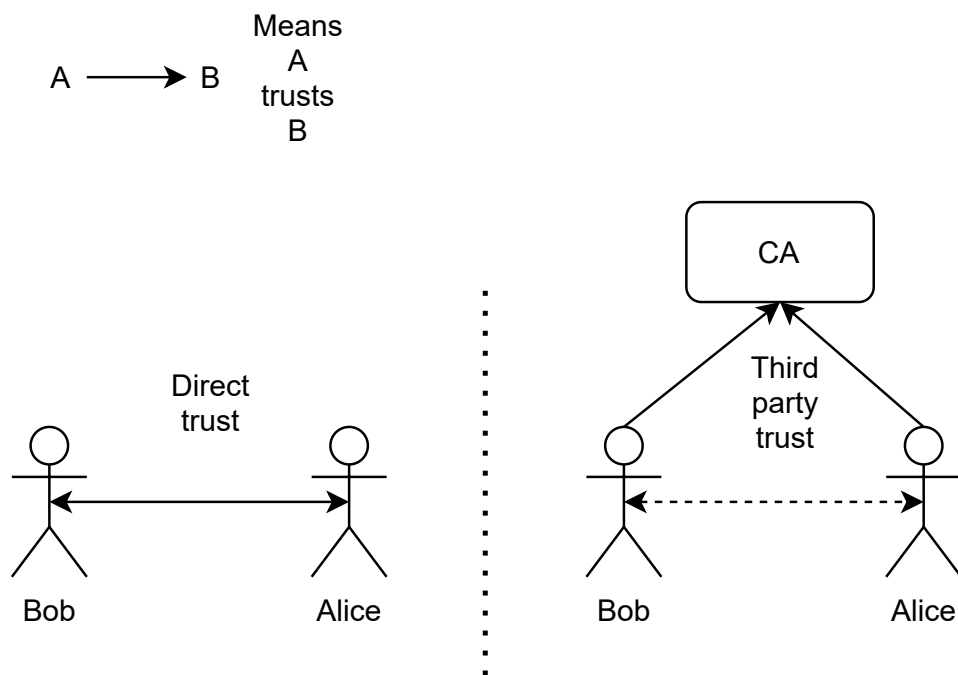


Figure 2.2: Direct trust vs. third party trust between individuals.

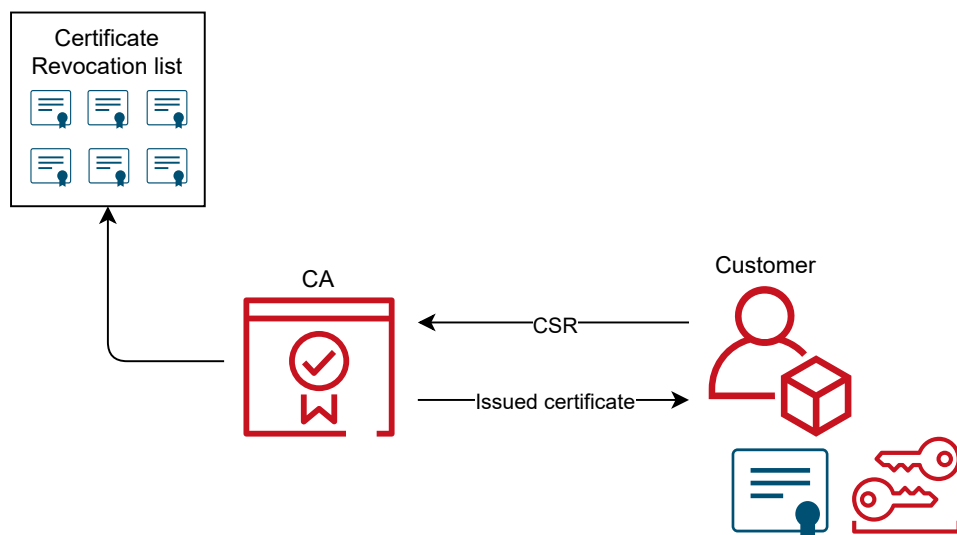


Figure 2.3: Flow of a certificate request in a C-PKI

## 2.2 Blockchain

Blockchain<sup>1</sup> is a data structure that was first used in 2008 with the introduction of Bitcoin[34]. The idealization of the blockchain was carried out aiming at a network without a central entity, where several participants, who run the same code, aim to reach consensus to maintain the integrity of the data recorded in the structure. In other terms, the blockchain is a decentralized append only list of records, also called blocks.

A blockchain block is composed of other fields on top of the data, which are:

- **Block number.** This is used to find the location of a given block into the chain.
- **Hash of the data.** This is a hash of the whole block data.
- **Nonce.** A nonce is a "number only used once", this number is included into the blocks in order to solve the computational puzzle that provides integrity to the blockchain. This concept is known as the Proof of Work which will be explained in more detail into the Section 2.2.1.
- **Hash pointer.** This is what links all blocks together and prevents a block from being tampered.

### 2.2.1 Proof-of-Work and Consensus

Contrary to certain beliefs[5], Proof of Work (PoW) is not what enables consensus into the blockchain. Consensus on the blockchain is obtained by taking the longest chain as the prevalent chain. In the case of a fork (i.e., the case where the chain separates into two separate chains), a peer will choose either one of the chains as the prevalent one and if the other one becomes longer (because a majority of nodes have chosen the other one), the peer will discard its chain and take the other one as the prevalent chain. That is why a transaction is considered accepted by the consensus into Bitcoin only when six blocks have been mined after the block containing that transaction.

PoW is used in blockchain in order to ensure integrity of the chain. The idea behind PoW is that no individual has enough computational power in order to falsify the chain. It works by asking to solve a computationally intensive puzzle in order to be able to submit a block onto the chain. The peers that compute these puzzles are called miners. The miners have a monetary incentive to mine these blocks because individuals pay money in order to have their transactions included into the next block. The miners may also be rewarded with money by the framework itself. There are

---

<sup>1</sup>If you want to play with this concept (i.e., Blockchain), there is a great browser-based blockchain created by Anders Brownworth at this URL: <https://andersbrownworth.com/blockchain/blockchain>

alternative methods to PoW, such as Proof of Stake<sup>2</sup> or Proof of Space<sup>3</sup>, that will not be addressed here.

### **2.2.2 Smart contract**

Smart contracts have been democratized in 2014 with the introduction of Ethereum[53]. A smart contract is a complete Turing machine hosted on the blockchain. To better illustrate the concept, you can view a smart contract as an automaton with memory. Therefore, a smart contract waits for events and reacts to them when it receives them.

The integrity of a smart contract and of its transactions is ensured by the underlying blockchain technology and therefore a smart contract inherits from the same properties as the blockchain. Nevertheless, smart contract transactions are different from regular transactions because they need to run code and therefore, the mining of these transactions must be incentivized in a different way. On top of the traditional transaction fee, smart contracts rely on "gas" which is an additional transaction fee paid to the miner for borrowing their computation ability on the blockchain.

### **2.2.3 Name-Value Storage**

NVS is basically a key-value store. This framework has been used by some blockchain technologies such as Namecoin[10], Emercoin [12] and so on.

---

<sup>2</sup><https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>

<sup>3</sup><https://en.bitcoinwiki.org/wiki/Proof-of-space>



## Chapter 3

# State of the Art

In the purpose of adequately depicting the related works, we decided to present all the different flavors available to set up a PKI targeted at IoT devices. The first section will be focused on the blockchain based solution. This section will talk about two different paradigms which are the Name-Value Storage (NVS) based solution represented by Certcoin[18] (and IoT-PKI [52]) and the other paradigm is represented by Scpki [4] which uses Smart contract. On the other hand, we have the CA based solutions that try to adapt the C-PKI infrastructure to match the IoT constraints.

### 3.1 Blockchain based solutions

#### 3.1.1 Certcoin and IoT-PKI

Both of these solutions are based on NVS blockchain, where an NVS is basically a key-value store. Despite using the same blockchain paradigm, those two architectures did not use the same technology. As a matter of fact, Certcoin is based on Namecoin[10] when IoT-PKI is based on Emercoin[12]. The idea behind these solutions is to store the public key as the value and use the domain name (or another identifier) as the key into this NVS. Nevertheless, this solution cannot be implemented naively because the whole blockchain needs to be stored in order to interact with it, which is unfeasible for IoT devices. Therefore, each of these solutions has a mechanism in place to avoid this problem. In the case of Certcoin, they proposed the use of accumulators[14] to lower the storage requirement. Where an accumulator is a digital entity that is used to test set membership. The accumulator holds tuples of the form  $(d, pk)$ , where  $d$  represents a domain and  $pk$  represents a public key. The accumulator is then used to see if  $pk$  is associated with  $d$ . Accumulators have the advantage of not needing much storage and can be easily used by computationally constrained devices (i.e., IoT devices). On the other hand, IoT-PKI use bigger devices to hold the chain and the IoT devices need to communicate with this intermediary instead of directly communicating with the blockchain technology.

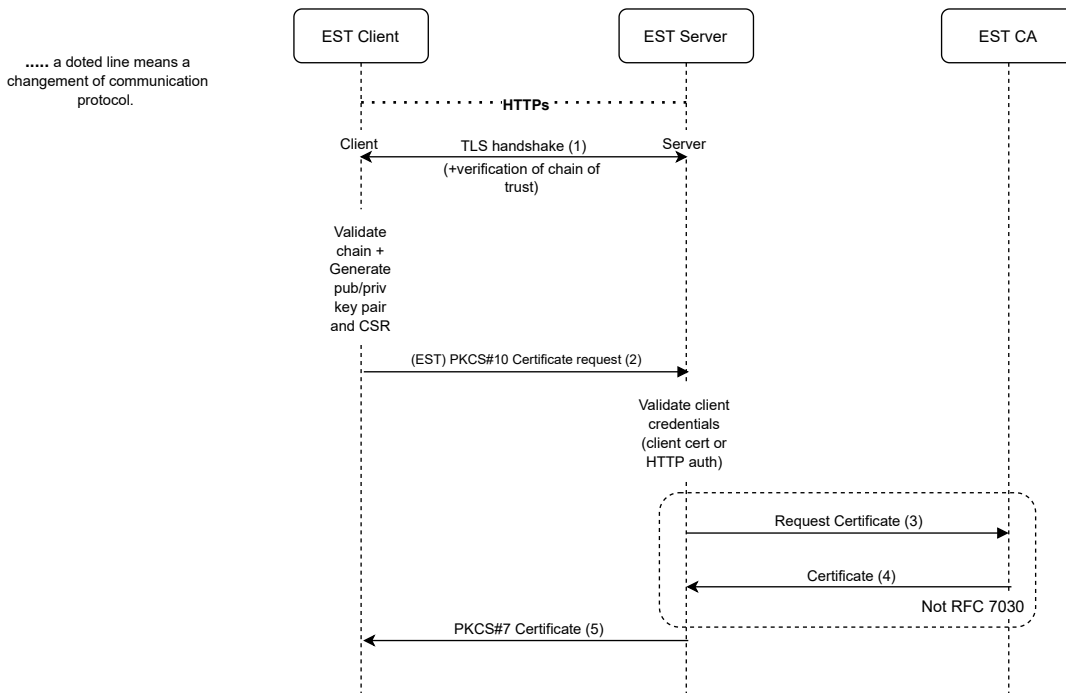


Figure 3.1: The certificate enrollment process using EST.

### 3.1.2 Scpki

In essence, Scpki is an implementation of the Web of trust using smart contract. The principle is that an individual in this framework is identified by its Ethereum address and this individual can publish attributes about itself on the system. Later, other individuals can sign these attributes to prove that the information is legitimate. Signature can also be revoked if needed. Therefore in this system, trust is enabled through reputation.

## 3.2 CA based solutions

### 3.2.1 EST over secure CoAP

"Enrollment over Secure Transport" (EST) is a protocol that automates the issuance of x.509 certificates for public key infrastructure (PKI) clients such as web servers, endpoint devices, and user identities, as well as the related certificates from a trusted Certificate Authority (CA).

In particular, EST over secure COAP[48] is a proposed adaptation of the EST over HTTPs protocol[1]. This protocol is still under development and should only be considered as such until it is included into the standard.

The purpose of this protocol is to port the state of the art C-PKI(i.e., EST over HTTPs) to more computationally constrained devices. As a reminder here is how EST over HTTPs works (The Figure 3.1 will be used as support):

1. A TLS session is created between the client and the server and the

chain of trust of the server certificate is verified by the client.

2. The client creates its key pair and a CSR before sending a PKCS#10 certificate request[35] to the server.
- 3, 4. The server requests and receives the certificate to the CA.
5. The server then sends the signed certificate to the client in PKCS#7 format[25], which the client can store on its device.

In addition to the certificate enrollment feature, an EST client may send a re-enrollment request to an EST server to update or rekey the current client certificate. Additionally, to facilitate other certificate activities, additional optional requests can be transported via EST.

The main differences between EST over HTTPs and EST over COAPs are the application and the security layer protocols. Apart from this, the two protocols are very similar.

### **3.2.2 PKI4IoT**

PKI4IoT[20] is a solution based on EST over COAPs with few additional features. The features include a new lightweight certificate format called XIOT which is backward compatible with x.509. PKI4IoT also proposes a way of dealing with initial authentication by using factory certificates.

PKI4IOT is a more polished version of EST over COAPs that has been tested on resource-constrained devices. Nevertheless, this solution is still under development as it is not part of the standard.

## **Part II**

# **The Framework**

## Chapter 4

# Security requirements and threat model

Security for such a framework is required as any correspondence between the individuals involved may be eavesdropped by an attacker. They may block or change messages, or store and replay any message sent, with the intention of masquerading as a trusted entity or to get hold of any secret message content in plain text.

On top of that, blockchain-based systems also have some shortcomings.

In more details, on the Figure 4.1, you can see the different families of attack that a blockchain technology should be ready to face and few examples for each family. The five families are:

- *Identity-based attacks*: are attacks where an attacker forges its identity to masquerade as another device or even create multiple illegitimate identities in the networks;
- *Manipulation-based attacks*: are attacks where the purpose is to manipulate the data to intentionally trigger external events that can be capitalized on;
- *Cryptanalytic attacks*: are a family of attacks where the purpose is to break the cryptographic algorithm and expose its keys. In this way, an adversary can sign unauthorized transactions and forge the valid signature of users;
- *Reputation-based attacks*: are attacks where an agent manipulates his reputation by changing it to a positive one;
- *Service-based attacks*: are attacks where attackers aim either to make the service unavailable or to make it behave differently from its specifications.

It is essential that all these attacks be taken into account when a new framework is developed on the blockchain. A discussion about how previous work dealt with those attacks can be found in [15] and a

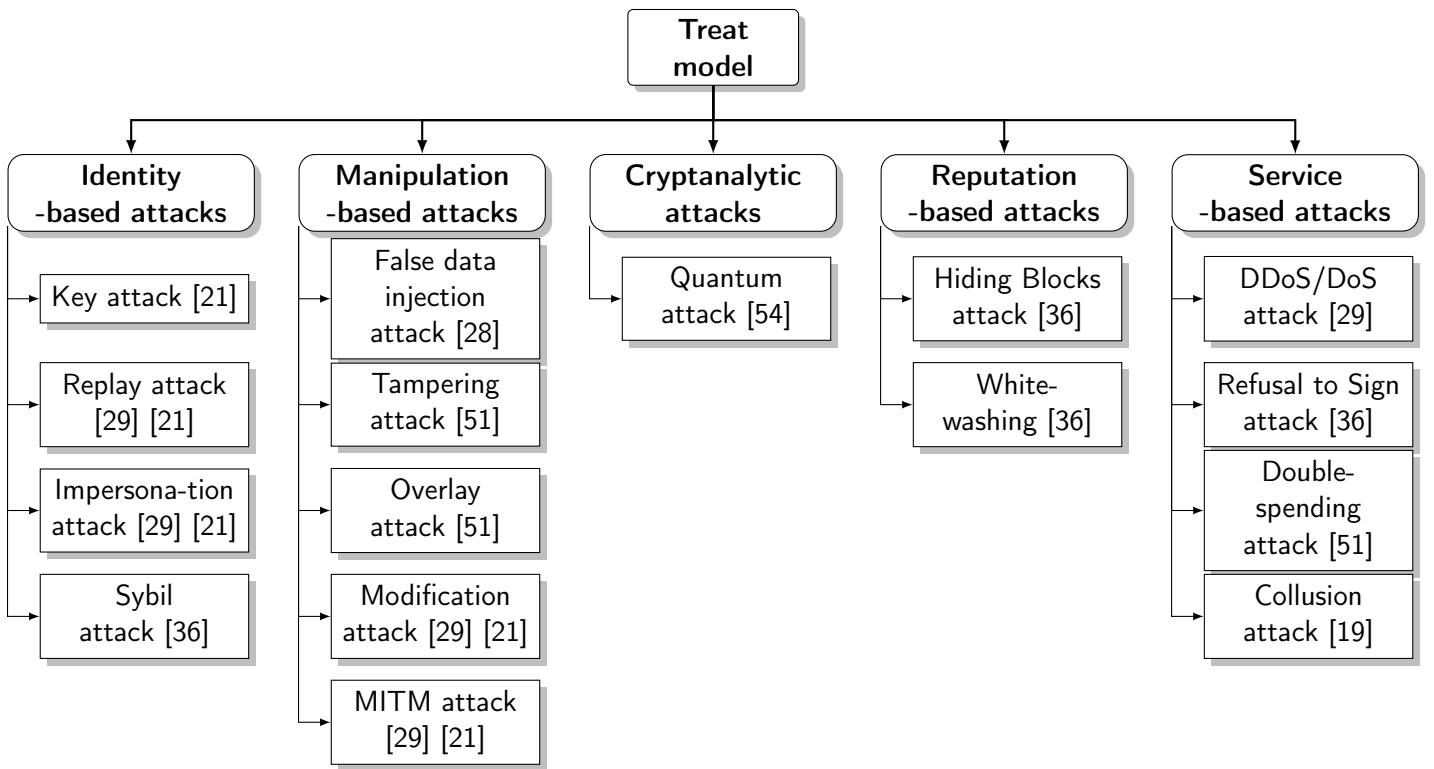


Figure 4.1: Classification of threat models for Blockchain. Source [15]

discussion about how the framework proposed deals with those attacks can be found in Section 7.

It is crucial that the proposed framework withstands these types of attacks and still offers authentication, confidentiality and integrity.

## Chapter 5

# Architecture

This chapter will display the architecture of the developed framework. In this architecture, we have three partakers which are the IoT device, the controller and the smart contract. Each of those components will be explained into their respective section.

### 5.1 IoT Controller

An IoT controller is a bigger appliance than the IoT device that will be used to compensate the lack of power of the IoT device. Unlike previous work[49], the controller will not fully manage the certificate of the device but rather will be there helping the device in the management of its certificate. The controller is also there to enable communication between the device and the smart contract. It is important to note that in this framework the controller will never be able to impersonate the device as the device remains the only one in control of its private key. To be more precise, the controller will be mainly responsible for verifying the validity of the certificate created by the device, converting the device data into a format understandable by the smart contract and providing randomness to the device (if needed) to create the key pair associated with its certificate.

#### 5.1.1 Nature of the controller

There are two possibilities for the implementation of the controller:

- A dedicated bridge/box that will deal with the certificates of several IoT devices located in different networks;
- A dedicated app on the user phone that will act as the controller. The problem with this type of controller is that it may not always be available and this can cause problems if the device needs to update its certificate while the controller is not available.

Due to the shortcoming of the last solution, we decided to opt for the first solution in this work.

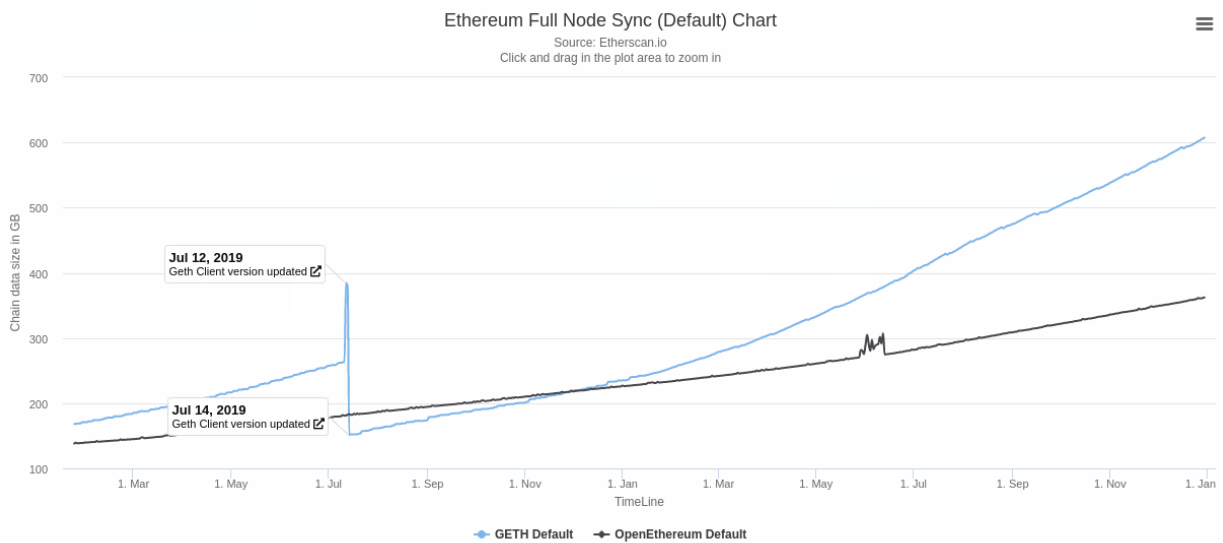


Figure 5.1: Chart of the storage needed to run a full node. Source: Etherscan.io

### 5.1.2 Ethereum nodes

In the purpose of communicating with the smart contract, the controller should run an Ethereum node or at least the controller should be able to communicate with a node. Nodes are clients of the blockchain technology. More precisely, a node is a device/program that communicates with the Ethereum network.

There are three types of nodes in the Ethereum blockchain:

- A **full node** has a copy of the entire Ethereum blockchain state and performs any transaction that is mined, taking up to 600GB<sup>1</sup> (See Figure 5.1) of storage and at least 4 GB of memory. A full node will take several hours to join the network and become fully synchronized. (Source: <https://ethereum.org/en/developers/docs/nodes-and-clients/>)
- A **light node** has only the minimal amount of state to make sense of things while communicating with full nodes. This type of node will only need a few hundred megabytes of storage and 128-512MB of memory. A light node's purpose is to be small enough to operate on a phone or embedded devices. A light node will almost immediately join the network and we can directly start using it. This is assuming that it can find a full node with a light node slot available. This is due to the fact that a light node relies on a trusted full node to gather the needed data. (Source: [38])
- An **archive node** is a full node that preserves the entire history of transactions (i.e. an archive node is a full node running in archive

<sup>1</sup><https://etherscan.io/chartsync/chaindefault>



mode). In the absence of archive mode, a full-node will prune created state to save disk space. This reduces the time it takes for a node to sync, as well as the cost of storage and computation. Because of the way Ethereum handles account and contract storage, only an archive node can serve API requests for RPC methods older than 128 blocks.

It seems obvious that the better suited type of node for the IoT controller is the light node. The light node provides the needed basic functionalities to the controller in order for it to be able to submit requests to the smart contract. The only drawback with this approach is that a light node relies on a full node and there may be no available full node for each controller. A solution to this drawback is for the controller manufacturer to run some full nodes or to pay cloud services to run full nodes for its controllers. This solution also brings some benefits: if the full node is hosted by the manufacturer, the manufacturer can have a better feeling of the trustworthiness of these nodes. This will provide a strong guarantee that the controller will always have a full node available.

## 5.2 IoT device

The IoT device is the central piece of this work since it will be the user of the framework and will use it to be authenticated. In order to distinguish each device from each other, the IoT devices should have a UUID. Ideally, this UUID should be of the version 4[37] where this version is the randomly generated UUID. The UUID should be generated with a cryptographically secure pseudorandom number generator in a trusted manufacturer environment, for example when installing the initial firmware to be used. These recommendations are there to avoid that an attacker can too easily guess the id of a device and set up a denial of service attack on this device (i.e. by recording this UUID on the smart contract before the device). This UUID will be used throughout the whole life cycle of the device to be identified. Due to the latter, the UUID should ideally be stored on a tamper proof memory. This UUID is what will later be authenticated by the framework.

### 5.2.1 Device life cycle

This section displays at high level the different states of the life cycle of the IoT device in this framework. The Figure 5.2 shows the different states and how these states interrelate with each other.

This life cycle can be decomposed into three phases. The first phase is the bootstrapping phase which can be defined as the process by which the state of an IoT device changes from not operational to operational. This phase can be seen as the initialization of the device. The generation of the UUID, the set-up of the device by the manufacturer and the initial boot up of the device are all regrouped in this phase.

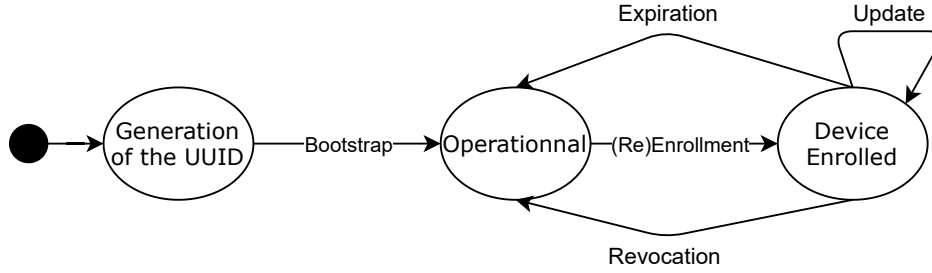


Figure 5.2: Life cycle of the IoT device.

The next phase is the enrollment phase by which the device will have its identity confirmed. This phase will make the device fully usable and authenticatable.

The last phase is the expiration/revocation phase. The last stage comes either when a certificate expires or when an administrator revokes the device certificate prior to the expiry date. A security infrastructure must cope with the revocation of trust if access to properties can no longer be given to a previously trusted individual for some reason.

This life cycle provides an overview of a certificate management framework for constrained devices.

## 5.2.2 Lasting certificate for the IoT device

In this framework, the lasting certificate of the IoT device will be a self-signed X.509[45] certificate where the identity of the device will be confirmed by the smart contract. In order to be complete, this certificate will contain the controller identifier. The controller identifier is a unique number that will be used similarly to identify the controller responsible of the device. This inclusion is used to provide a proof of the binding between the controller and the device. This binding is necessary as the controller will be the one responsible for the update and the revocation of the device certificate. As the certificate is self-signed, the IoT device also confirms the binding.

We have to take into consideration that the controller may disappear. Therefore, the controller should implement some kind of key sharding<sup>2</sup> where the controller stores its shards into separate locations in order to be able to be replaced by a new controller in the case where this controller disappears or is broken.

We must nevertheless be careful that X.509 certificates may be considered heavy for some IoT devices[17]. If not dealt carefully, this may cause significant overhead when this solution will be ported to more resource-constrained IoT devices. Previous work has shown that the X.509 format can, however, be used on small devices, but naively using existing X.509 certificates may cause problems [40]. Hence in the case of very small devices, a new lightweight format that does not break X.509 compatibility

<sup>2</sup>This is a process where a private key is split into pieces (i.e., shards)

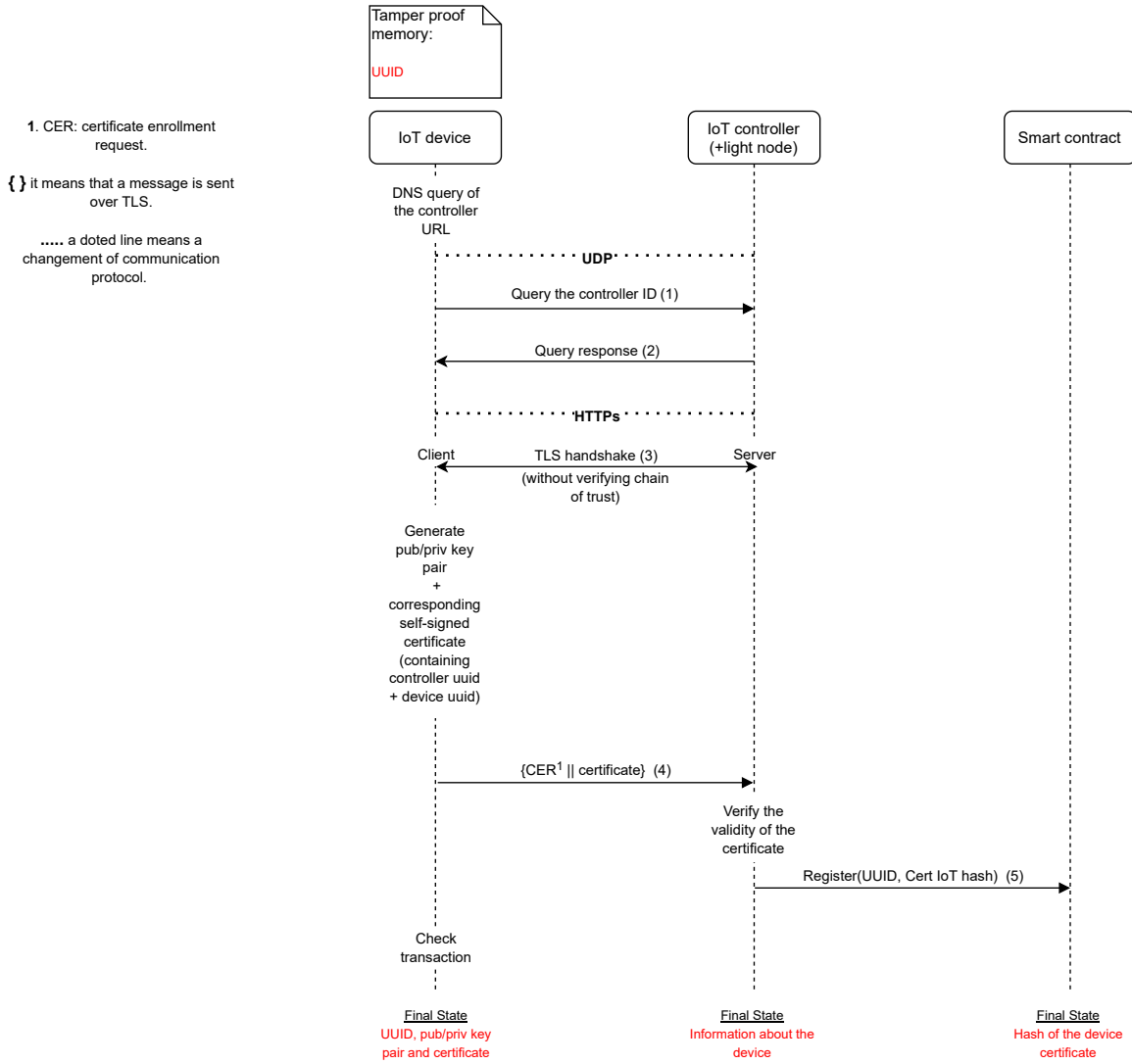


Figure 5.3: Device enrollment into the PKI (explanation can be found in the Section 5.3 : Device enrollment )

may be needed. XIOT [20] is such a format. The paper presenting XIOT claims that the certificate size is reduced by more than 50%. Therefore, XIOT is a viable alternative to X.509 if the device lacks memory.

### 5.3 Device enrollment

The enrollment of the device can be seen step by step on the Figure 5.3. Following, we have an overview of what is performed at each step:

1. The first thing that a device needs to do is to find the ID of its controller. As a reminder, the device needs the controller ID to generate its certificate as explained in Section 5.2.2. We decided that the device should be the one to begin the transaction for the simple

reason that there can be several candidate controllers in the network. The fact that the device begins the transaction allows it to choose between controllers.

2. Once the device receives the ID, it will generate its certificate with the corresponding public/private key pair. If the device is not random enough<sup>3</sup> to generate the asymmetric key pairs associated with its new certificate, it is possible to add a step between the steps 3 and 4 where the controller sends a random number that will be used as a seed by the device in order to gather enough entropy. It is important that the device generates its own certificate to avoid impersonation by a malicious controller.
3. When the device has chosen a controller, they need to perform a TLS handshake [42], where the client is the device and the server is the controller. In this step, the controller will also verify the validity of the device certificate.
4. The certificate enrollment request displays the request of the device. The certificate sent at this step is the newly created certificate of the device.
5. If the certificate is valid, the certificate of the IoT device is forwarded to the smart contract by calling the smart contract function *Register* which is described in detail into the Section 5.7.<sup>4</sup>
6. As soon as the transaction is mined on the blockchain, the device can check that its data appears on the transaction ledger.

## 5.4 Device certificate update

Device certificate update in a PKI framework is primordial to avoid overuse of the related key pair. The longer a key pair is used, the further it increases the attack surface. It makes the certificates and keys an attractive target for malicious actors attempting to hack a credential, as it allows them access for longer periods of time. Recommendation on cryptoperiods<sup>5</sup> from the NIST can be found in Figure 5.5. Due to those recommendation, we decided to add to our framework an update feature.

This update protocol can be visualized in Figure 5.4. The different steps of this protocol are:

---

<sup>3</sup>Embedded devices are typically cheap and without security hardware, such as a Trusted Platform Module and a Physical Unclonable Function. Furthermore, they often cannot gather enough entropy due to short boot times. Therefore, their random number generators may not be cryptographically secure.

<sup>4</sup>NB: calling a smart contract function is actually a multicast message sent to the full nodes of the blockchain technology (i.e., Ethereum).

<sup>5</sup>A cryptoperiod (sometimes called a key lifetime or a validity period) is a specific time span during which a cryptographic key setting remains in effect.

1. CUR: certificate update request.

{ } it means that a message is sent over TLS.

..... a dotted line means a changement of communication protocol.

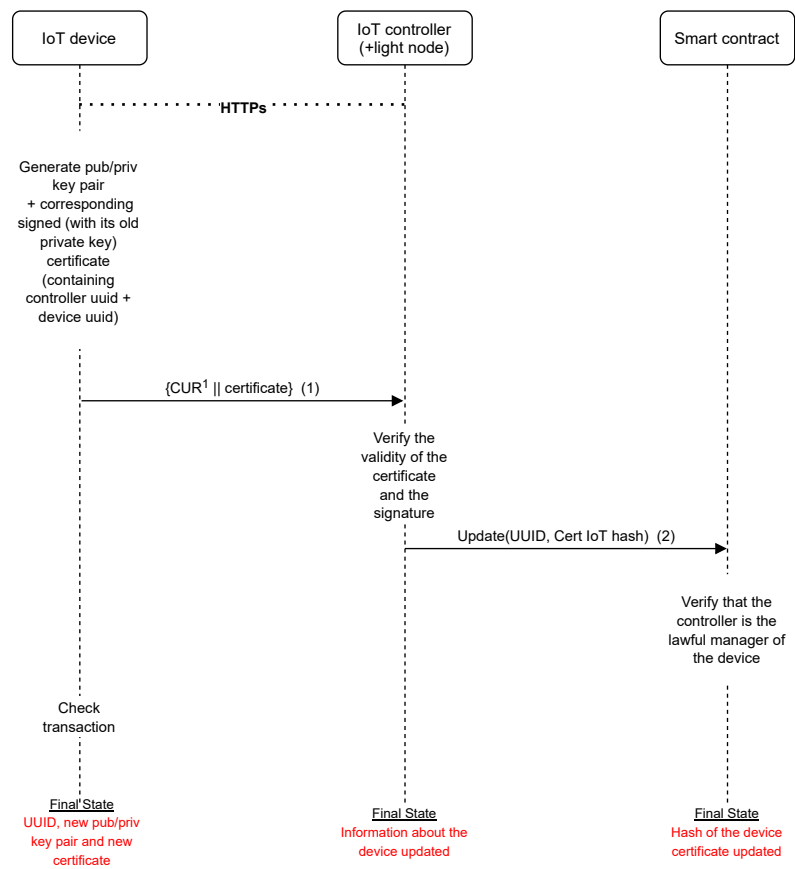


Figure 5.4: Device key update (explanation can be found in the Section 5.4)

Key Type <i>Move the cursor over a type for description</i>	Cryptoperiod	
	Originator Usage Period (OUP)	Recipient Usage Period
Private Signature Key	1-3 years	-
Public Signature Key	Several years (depends on key size)	
Symmetric Authentication Key	≤ 2 years	≤ OUP + 3 years
Private Authentication Key	1-2 years	
Public Authentication Key	1-2 years	
Symmetric Data Encryption Key	≤ 2 years	≤ OUP + 3 years
Symmetric Key Wrapping Key	≤ 2 years	≤ OUP + 3 years
Symmetric RBG keys	See SP 800-90	-
Symmetric Master Key	About 1 year	-
Private Key Transport Key	≤ 2 years <sup>(1)</sup>	
Public Key Transport Key	1-2 years	
Symmetric Key Agreement Key	1-2 years <sup>(2)</sup>	
Private Static Key Agreement Key	1-2 years <sup>(3)</sup>	
Public Static Key Agreement Key	1-2 years	
Private Ephemeral Key Agreement Key	One key agreement transaction	
Public Ephemeral Key Agreement Key	One key agreement transaction	
Symmetric Authorization Key	≤ 2 years	
Private Authorization Key	≤ 2 years	
Public Authorization Key	≤ 2 years	

Figure 5.5: Suggested cryptoperiods for key type. Source: section 5.3.6 in the report [3]

1. As the device already has the controller ID from its original registration, it can directly send the certificate to the controller. Upon reception, the controller will check the validity of the certificate and verify that the certificate is signed with the old private key of the device. This last mechanism is used to avoid impersonation of the device. If the controller is offline, the device should back off and try later. A better solution would be to implement some kind of instant messaging technology where messages are stored while the controller is offline and when the controller goes back up, it can retrieve the messages sent.
2. If the certificate is valid, the hash of the certificate will be sent to the smart contract. The smart contract will then check if the controller is the lawful manager of the device by comparing the sending Ethereum address (i.e., the Ethereum address of the controller), to the one stored with the certificate data (i.e., the owner address stored in the smart contract). If the check succeeds, the hash of the certificate will be updated into the smart contract.

## 5.5 Device certificate revocation

Certificate revocation is an essential PKI component as the certificate can be compromised. In that case, action should be taken immediately after the discovery in order to avoid as much damage as possible. As a matter of

fact, a compromised certificate can lead to a leak of precious information and to impersonation. Revocation can also be useful to deal with access control.

In this framework, device certification revocation is managed by a simple function call on the smart contract. If we want to revoke a device certificate, the controller simply needs to call the `Revoke` (more information can be found in Section 5.7.3) function on the smart contract. As for the update function, this function can only be called by the lawful manager (i.e., the responsible controller) of the device. This constraint ensures that no attacker can revoke a valid certificate.

## 5.6 Device lookup

A device can lookup the data stored on the smart contract by making an HTTPs GET request on a controller where the path of this GET request is the UUID of the targeted device (i.e., from which you want the data) and the controller will just forward to the device the output of the "lookup" function from the smart contract.

## 5.7 Smart contract

The smart contract is a central piece of this framework as it acts as the trust store for this PKI. The role of the smart contract is mainly to maintain an up to date status of all device certificates.

### 5.7.1 Smart contract vs. NVS

Lots of decentralized PKI in the literature use Name-Value Storage (NVS)[52] because it makes the implementation trivial (you store the certificate as value (or hash) and the id is the key). But despite being easy to use, it is hard to verify what is pushed on the blockchain with an NVS. This is why we decided to use smart contracts instead. Smart contract permits to verify the certificate before being pushed on the chain. For example, the contract can verify that the key pair associated with the certificate is strong enough, that the expiration date of the certificate is not too long, and so on. Furthermore, smart contract makes the upgrade of the system far easier as new functionalities can directly be added on the smart contract. In the case of NVS, new functionalities must be implemented on the client side (i.e. the IoT device) which makes the update more difficult as NVS lacks of flexibility and IoT devices are computationally restrained.

### 5.7.2 Data stored on the blockchain

The data of the device certificates is stored in a private hash map where the key is the device UUID and the value is a structure (can be seen on Figure 5.6) that represents this data. A smart contract private field acts the same

way as a private field in object oriented classes, therefore only the smart contract can access this hash map.

The data stored into the hash map is shown on Figure 5.6 where the fields are:

- Hash of the IoT certificate: It is the hash of the device certificate. It can be requested by anybody in order to verify the integrity of a received certificate;
- Controller Ethereum address: This is used to identify the manager of the device and therefore authorize or not an update and/or a revocation of the device certificate;
- Status: The status of the certificate is used to state if the certificate is either valid or revoked.

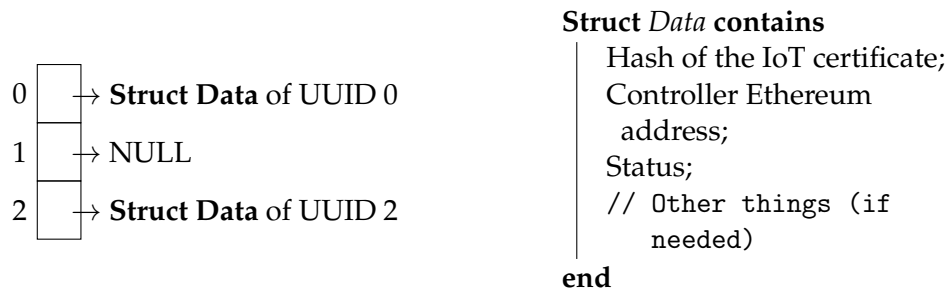


Figure 5.6: Structure of the data stored on the smart contract hash map

### 5.7.3 Setter/Getter functions

(string, int) **Lookup**(UUID)

This function can be used by anybody to request the hash of a certificate identified by the UUID given as a parameter. This function will return the status of the certificate with the certificate hash.

void **Register**(UUID, hash of the device certificate)

This function is used to register the lasting certificate of the IoT device on the smart contract. This function will store the UUID and the hash into its private map.

void **Update**(UUID, hash of the device certificate)



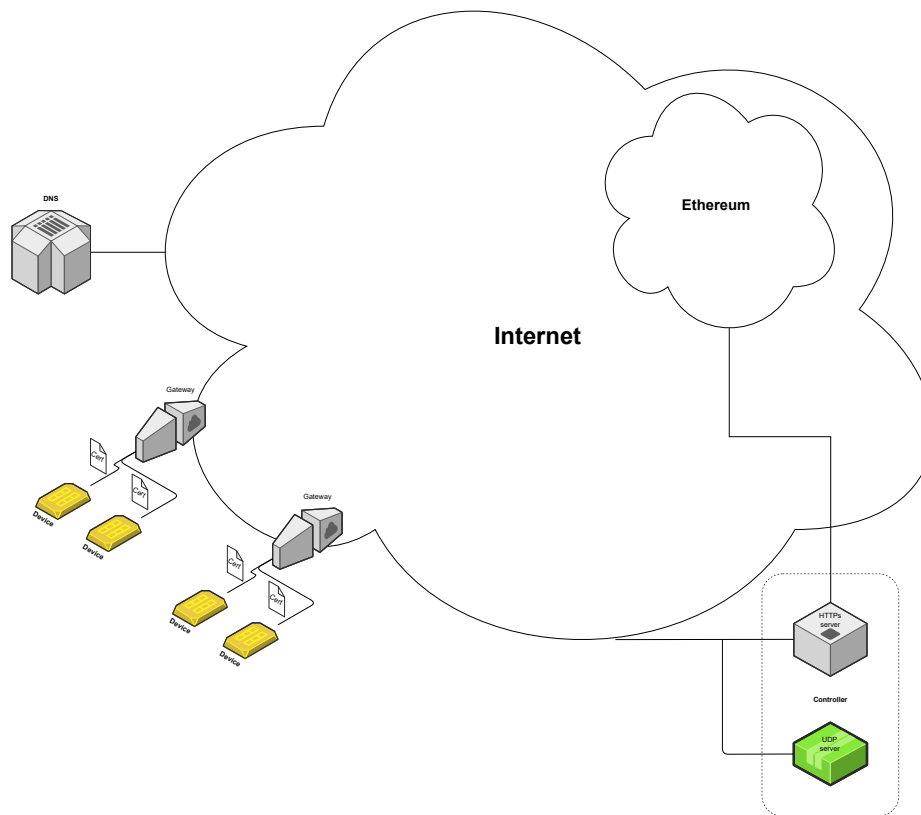


Figure 5.7: Overview of the whole architecture

This function is called when the controller needs to make a key rotation for the device. If the message sender address corresponds to the manager Ethereum address stored with this UUID, the smart contract will replace the certificate hash with this new hash into its hash map.

```
void Revoke(UUID)
```

This function will change the status of the UUID from valid to revoked. As for the update function, this function will only be executed if the message sender address matches the manager address.

## 5.8 Overall view of the architecture

The overall architecture can be seen on Figure 5.7. Where the different components are:

- The DNS server is used by the devices to request the controller IP;
- The devices located in different networks;

- The controller with its two components (UDP server and the HTTPs server). The first one is used to request the controller id and the second one is used to receive the device certificate and communicates with Ethereum.

## 5.9 Possible improvements

Here is a list of improvements that can be made to this framework:

- Develop an instant messaging system to log requests made on offline controllers. Therefore, a controller will be able to see what requests it received while offline;
- Put in place key sharding<sup>6</sup> and store the key shards into different locations to be able to recover lost or broken controllers without losing the certificate binding between the device and the controller;
- Make the framework more lightweight by switching from HTTPs to COAPs;
- Switch from simple DNS to DNSsec in order to be certain to have a legitimate controller (More details into Section 9.3);
- Switch from classical device UUID to secure UUID to enable initial authentication of the device. Where a secure UUID is a simple UUID with a private key linked to the UUID (the UUID acts as the public key and have a private key linked to it to prove UUID ownership).

---

<sup>6</sup>This is a process where a private key is split into pieces (i.e., shards)

**Part III**

**Framework analysis**

## Chapter 6

# Scalability of the framework

This framework mainly relies on the blockchain and therefore inherits from the blockchain problems. In the purpose of adequately discussing the biggest blockchain flaw, we have to take a step back and define the trilemma that each system has to face in one way or another.

### 6.1 The DCS trilemma

The Decentralized Consensus Scale (DCS) trilemma[47], which is also called the blockchain trilemma, is a trilemma that states that a system can only have two of the following properties at the same time. Where the three properties are:

1. **Decentralized.** In systems theory, a decentralized system is one in which lower-level elements operate on local knowledge to achieve global objectives. In such system, there is no single point of failure or control.
2. **Consensus.** It means that different entities in a peer to peer system communicate with each other (through a consensus algorithm) in order to update the state of the system in a consistent manner between each other.
3. **Scale.** It ensures that the system can handle the transactional demands of any rival system that provides the same service to the same random group of consumers all over the world ("at scale").

This trilemma can be visualized on Figure 6.2 (a system must choose an edge of the triangle). The proof of this claim (i.e, a system can only have two of these properties at the same time) can be seen in the paper [47].

### 6.2 Implication of the DCS trilemma on the framework

As the system used in this framework is Ethereum and this technology chooses "Consensus" and "Decentralization" over "Scale". This means that if

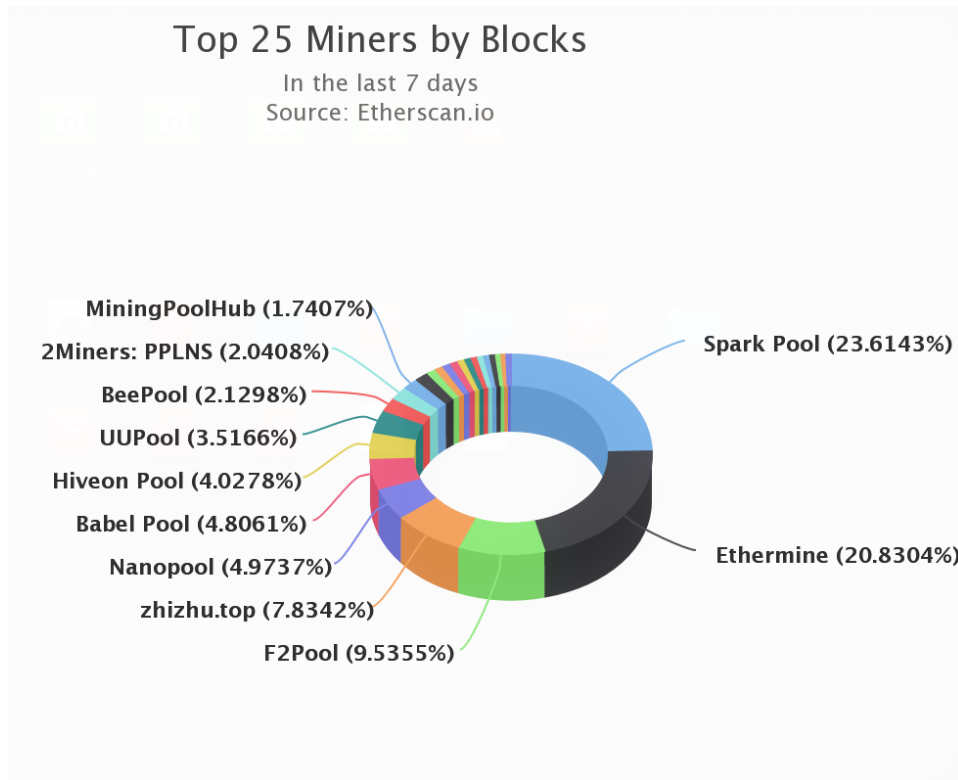


Figure 6.1: Pie chart showing the top 25 miners of the Ethereum Blockchain.  
Source: Etherscan.io

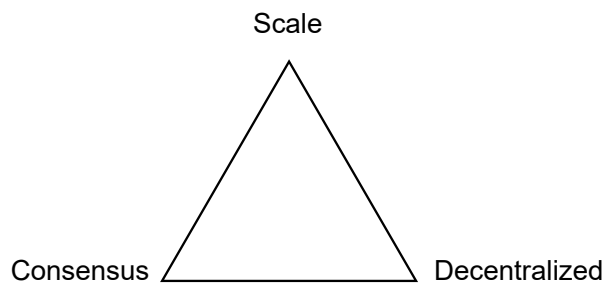


Figure 6.2: Illustration of the DCS trilemma.

a similar framework was used where we change the smart contract in favor of a centralized system, will always be faster than the proposed framework. Nevertheless, the implication of using a centralized system is that we must have blind trust in that system (i.e., we lose decentralization).

On top of that, we can already see that as Ethereum becomes bigger and bigger, it tends toward a centralized system. As a matter of fact, the bigger the blockchain becomes the more resources are needed to be part of the consensus. This has the impact of reducing the number of parties participating into the consensus (i.e., tends toward a centralized system). The evolution of the blockchain size can be seen on Figure 5.1.

Miners that do not have enough resources to store the whole blockchain collude into pools where a master node dictates to those miners what the state of the chain is. We can see those pools on the Figure 6.1. In the past, we have seen the impact that a single mining pool had on the Bitcoin framework when one of the biggest mining farms was lost due to flooding in China in August 2020[41]. The impact of this loss was a 20% hashrate Losses for Chinese Bitcoin Miners. The fact that this farm had such a big impact on the Bitcoin PoW is one more hint that these systems tends toward centralization at scale.

### 6.3 Solutions to that limitation

Solutions to these problems are already in development and hopefully will be released before damage is done. As a matter of fact, the next version of Ethereum (i.e., Ethereum 2.0) will have features that mitigate that problem. Ethereum 2.0 will switch to a Proof of Stake (PoS) blockchain and will use sharding. In a PoS system, block producers are in charge of checking transactions. Block producers are selected at random, but their chances of being one rise in direct proportion to the amount of tokens they own. Block producers are required to bind a number of tokens to their decisions while submitting a block, and they can be penalized if they act maliciously. The purpose of switching from PoW to PoS is to remove the need of having massive resources to participate in consensus. We can already see that this is only a mitigation as few people can have most of the tokens, which will have the consequence of also centralizing the system. Sharding<sup>1</sup> is the principle of cutting the main chain into smaller independent chains. This will have the consequence of eluding the scale factor of the trilemma. Ethereum 2.0 will be divided into 64 shards.

---

<sup>1</sup>This is system sharding which is different from key sharding explained in the previous chapter.

## Chapter 7

# Security assessment of the developed framework

### 7.1 Study of the feasibility and security of the enrollment stage:

To better illustrate our study, we will use as support the Figure 5.3. We can see that for the first two steps that the messages are sent over UDP which means that there is no integrity, no confidentiality and no authentication. On the point of confidentiality, since everyone can request the id, there is no need for confidentiality.

The question becomes a little bit more tricky on the topic of integrity and message origin. The only thing that can happen in the case where the device receives a wrong ID is that the controller will not respond later to the device request and therefore will cause a denial of service on the device. In this work, we consider that this is an acceptable shortcoming compared to the load that needs to be added in order to provide integrity on these steps.

For all the other communication between the device and the controller, a digital signature is included. This ensures to both parties the integrity and source of the messages exchanged. Confidentiality is achieved by encrypting each message with the symmetric key exchanged in the TLS handshake.

The main purpose of TLS is to provide privacy, data integrity and authentication for communication between a client and a server. A security analysis of this protocol can be seen in more detail in the paper[33].

The problem with using TLS for IoT devices is that some smaller devices may lack of computational power. The solution to this problem resides in the usage of lightweight implementations of TLS or even using DTLS. Currently, there are multiple lightweight implementations of the TLS protocol available to meet the constraints of low-powered IoT devices. The open-source DTLS Toolkit <sup>1</sup> (formerly MatrixSSL) can be configured to

---

<sup>1</sup><https://www.rambus.com/security/software-protocols/secure-communication-toolkits/tls-toolkit/>

a code footprint of only 66KB, and even smaller footprints are possible with manual optimization. wolfSSL <sup>2</sup>, another open-source SSL/TLS library, advertises a minimum footprint size of 20-100KB, and runtime memory usage of 1-36KB. Clearly, these numbers are already achievable even for a device with very modest specifications such as the Firefly<sup>3</sup>, and we can expect to see further optimization of software combined with increased power at lower cost for embedded devices going forward.

Furthermore, a recent study [43] has judged that IoT devices can benefit from the latest version of TLS/DTLS without the need to upgrade the hardware. But using (D)TLS protocol improvements comes at a price as we can see from their experiments. Compared to version 1.2, their measurements indicate 20% code footprint increase with TLS/DTLS 1.3. Further measurements on the RAM indicate no significant impact on stack requirements, and limited impact on peak heap footprint (max 25% increase for some configurations in one implementation, and for the other implementation, even 30% decrease, for Elliptic Curve Diffie-Hellman Ephemeral configurations).

For function call to the smart contract, the integrity, the authentication and basic confidentiality is provided by Ethereum. Ethereum relies on the controller wallet<sup>4</sup> to identify the controller and to sign messages. On the topic of confidentiality, Ethereum provides it by only sending byte code to the smart contract. This form of confidentiality is not optimal as the byte code can still be reverse engineered. On top of that, Ethereum transactions are also protected from replay attack by using a transaction count in each transaction. It should also be noted that every full node verifies the execution of the smart contract function, therefore Ethereum also protects the smart contract code and data integrity.

## **7.2 Study of the feasibility and security of the update and revocation stages:**

For these stages, the communication between the controller and the device is performed on top of TLS and therefore receives the same benefits as for the enrollment phase. The communication between the controller and the smart contract is also protected by Ethereum on these stages. The difference in terms of security on these stages is that we need to verify in the smart contract that the update/revoke request originates from the controller that manages the device. This was done by checking that the Ethereum sender address is the same one used to register the device. In practice this is done by storing the manager Ethereum address with the certificate hash on the smart contract. On top of that, the updated certificate of the device is signed with the previous key (of the device) in order to avoid device impersonation.

---

<sup>2</sup><https://www.wolfssl.com/products/wolfssl/>

<sup>3</sup><https://github.com/Zolertia/Resources/wiki/Firefly>

<sup>4</sup>As a reminder, an Ethereum wallet is a public/private key pair used to sign transactions



### 7.3 Security limitation

In the current framework, the controller identity is not verified. An attacker could therefore trick a device to communicate with him instead of a legitimate controller. The consequence will be that the attacker will be able to register the device UUID on its name, which will lead to a denial of service on the device as the device will never be able to register its certificate with its UUID (because the slot on the smart contract is already taken by the attacker). However, the device can easily find this out as the device can request the smart contract data to another controller and see that the data is not what it published. This problem can easily be solved by using DNSsec instead of classical DNS to gather the controller IP.

**Part IV**

**The Proof of Concept**

# Chapter 8

## Technologies

For the development part of the Proof of Concept (POC), we have decided to use the Solidity<sup>1</sup> language with the public Ethereum blockchain. Solidity is an object-oriented, high-level language for implementing smart contracts. This language is statically typed, supports inheritance, libraries and complex user-defined types among other features. This means that complex programs can be implemented with this language. For example, Solidity has already been used to create contracts for uses such as voting, crowdfunding, blind auctions, and multi-signature wallets[44][50][30]. Those smart contracts are run on the Ethereum Virtual Machine (EVM) (More information in Section 8.2) and are accessed by contract address<sup>2</sup> which ensures code immutability once published on to the Ethereum blockchain.

### 8.1 Why solidity and public Ethereum

We have decided to use Solidity simply because it is the most used smart contract language[16] and therefore also the one with the most support. One downside of this pick is that Solidity is a new programming language and therefore Solidity will undoubtedly undergo major changes and newer versions of this language are not always backward compatible.

As for the choice of the Ethereum public blockchain, this is mainly due to the fact that this one of the oldest blockchains (July 30, 2015) and therefore most of the early stages vulnerabilities have already been patched. Public Ethereum is inevitably faster due to more participation, something that just cannot be said for other distributed ledger solutions. Public Ethereum is also used by lots of developers and by big companies which means that if vulnerabilities are found, they will be quickly patched.

---

<sup>1</sup><https://docs.soliditylang.org/en/latest/>

<sup>2</sup>An Ethereum smart contract address is a unique key (public key) used to invoke that contract on the blockchain network.

## 8.2 Ethereum Virtual Machine

Ethereum virtual machine, in essence, provides a layer of abstraction between the running code and the running system. This layer is needed to increase device portability and to ensure that applications are isolated from one another and from their host. Basically, the EVM translates Ethereum `op_code` into runnable instructions on the host machine.

## Chapter 9

# Implementation

In order to prove the feasibility of our framework, we decided to develop a Proof of Concept (PoC). The construction of the PoC is subdivided into two different parts. The first part is the interaction between the IoT device and its controller. This first part can be further divided into smaller parts which are :

- Device/Controller communications;
- Construction of the custom X.509 certificates;
- Device/Controller discovery.

As for the second part, it concerns the interaction between the controller and the smart contract. Each part uses its own set of protocols and operates separately.

### 9.1 Device/controller interactions

This section aims at explaining how the controller and the device communicate between each other.

#### 9.1.1 Device/Controller discovery

In this framework Device/Controller do not know each other at first. Therefore there should be a way for each other to automatically discover each other. In this purpose, we decided to use Domain Name System (DNS). This is done by creating a custom URL for the controllers that each device could access. We also tried alternatives such as using the Simple Service Discovery Protocol (SSDP). Essentially, this protocol permits the controllers to emit a beacon containing their respective ID and IP address. Once the beacons are set up, the devices can listen to the beacons and choose a controller to operate with. This solution had the advantage that it provides the id of the controller in the beacon but the big drawback of SSDP is that it only works in local networks and therefore it makes the need of a controller arise in each local network where a device is present.

The DNS solution also has a drawback which is that the id of the controller is not communicated with the IP address. Nevertheless, we decided to still use DNS due to its massive advantage: a controller can be responsible for several subnets. To solve the drawback of DNS, we decided to create a UDP server that the devices can query to get the ID of the controller.

### **9.1.2 Device/Controller communications**

The communication between the device and the controller is performed over the Hypertext Transfer Protocol (HTTP) with TLS. As a reminder, within this framework there is no need for authentication between the device and the controller because the controller can be authenticated later by the smart contract if needed or the device can use DNSsec to acquire the IP address of the controller. In practice, this is done by using a self-signed certificate for the controller (server) and no certificate for the device (client) into the TLS handshake.

These two entities communicate mostly to transfer certificates from the device (client) to the controller (server). This transfer is performed by a HTTP PUT method.

An alternative to HTTP for smaller devices has also been explored, this alternative used the Constrained Application Protocol (COAP). The communication process for the two protocols was essentially the same. The difference was that with COAP, we had to deal ourselves with the packet fragmentation. This limitation is what motivated the use of HTTP instead of COAP.

Indeed, a COAP packet should fit in exactly one IP packet. This assumption is made by the COAP protocol in order to make it lightweight. The consequence of this is that the certificate should be cut into pieces and sent over several PUT messages. The certificate should then be reconstructed at the receiving end. In order to permit that, the certificate is first cut into pieces and a header is added to each piece. This header contains a sequence number and the certificate size. A more general solution could implement this IETF draft [7]. It extends core COAP with new options for transferring multiple blocks of information from a resource representation in multiple request-response pair. Therefore COAP may become better suited in this case if the draft is later included into the standard.

### **9.1.3 Construction of the custom X.509 certificates**

In the purpose of binding the device with the controller, we decided to use custom certificates for the devices.

As part of the standard, X.509 v3 permits to add custom fields to the certificate. In this case, the custom field added is the unique identifier of the controller responsible of the device. This solution has been put into place by writing an openssl configuration file that must be used to generate the custom certificates.

## 9.2 Smart contract

The smart contract is the core of this framework, it acts as the trusted authority. The contract provides a secure storage to store information about the devices.

### 9.2.1 Structure of the contract

The trusted store in the smart contract is in practice implemented by a private map that is only accessible by the contract functions that are later described in this section. The mapping in this contract is a mapping between a device UUID and the device information. The UUID is represented here by a uint256 to mitigate collision and thus provide faster access to the mapping and also to limit as much as possible the impact of a corrupted controller<sup>1</sup>.

To modify/access the smart contract data, there are four public functions which are:

1. lookup: used to request the hash and the status of a device certificate associated with a given UUID;
2. register: used to register a device into the system;
3. update: used to update the information of the device on the system;
4. revoke: used to revoke the device certificate.

The lookup function is a little bit different from the three other functions as it is the only function that does not modify the smart contract data. Therefore, this function has been implemented as a "view" function and is the only function that does not cost any gas.

The update and revoke functions require that the message sender (identified by its Ethereum address) is the same as the one that previously registered the given device. This constraint is what ensures identity retention for the registered device.

### 9.2.2 Deployment

In order to test the framework, the contract had to be deployed. For the deployment two steps were taken, first the contract was deployed locally in order to quickly debug the code and to make basic unit tests on the different functions. This allowed us to verify that the contract had indeed the expected behavior. Unfortunately, this deployment solution has lots of drawbacks.

First of all, a local deployment does not reflect the latency that can be encountered on the main net (Ethereum) and local deployment makes you

---

<sup>1</sup>A corrupted controller could in theory register a major part of the identifiers available in order to make the system unusable (DoS attack) but this type of attack is mitigated by using uint256 to represent the UUID because the amount of gas (and thus money) to implement such an attack would be huge

able to run anything without any constraint (in terms of gas for example) which is not true on the main net. This is why the second step was to deploy the smart contract on a test net. As a reminder, a test net is a real network for test purpose similar to the Ethereum main net but with worthless Ethers. The chosen test net for the deployment is Kovan<sup>2</sup>. The deployed contract can be seen on Kovan Etherscan at this address<sup>3</sup>. Kovan is the best way to get a simulation as close as possible to the real deployment[6].

Another tool, Infura<sup>4</sup>, was used to solve one of the biggest problems we had encountered while deploying on Kovan. The problem is that a light node needs to be linked to a full node to interact with Ethereum but from time to time there are no more light node slots available (on the full nodes) and therefore the light node is not able to interact with the smart contract, in this case. The solution was to use Infura which is a full node provider that ensures accessibility of the smart contract. As a matter of fact, without a full node provider, we cannot be certain that there will be a light node slot for our controller available on a full node. In conclusion, Infura ensures that we can always access the smart contract provided that the number of daily transactions allowed by the free version is not exceeded. Moreover, Infura does not question the validity of the test since it only manages access to the smart contract and nothing else. So Infura will not be able to impersonate a controller or modify messages as they are signed by the sender (i.e., the controller).

### 9.3 Improvements

Even if this proof of concept is ready to be deployed, the current version of it is far from perfect. As a matter of fact, lots of things can still be improved. Here are few things that can be improved (on top of the framework improvements listed in Section 5.9):

**Code integrity** In the current version, the owner of smart contract has no proof that the code executed is really the code that has been published. This verification can be done in the future by adding the smart contract code to the Etherscan's token information. Etherscan will then verify that the contract code is exactly what is being deployed onto the blockchain and this will also allow the public to audit and read the contract code.

**Controller identity verification** Currently, the controller identity is never verified which might be a problem as an attacker can act as a controller and put in place a DOS attack on a device. Some solutions were explored to circumvent this problem. The easiest solution is for the device to use DNSsec instead of DNS to get a controller's IP. Another solution would be to authenticate the controller in the scope of the smart contract. The most obvious way of doing that is by verifying the RSA signature of

---

<sup>2</sup><https://kovan-testnet.github.io/website/>

<sup>3</sup><https://kovan.etherscan.io/address/0x2dde419f1369a4ef6b6408245479589cab4ff974>

<sup>4</sup><https://infura.io/>



the controller's certificate into the smart contract. Unfortunately at the moment, the RSA library has still to be implemented in the standard. Another solution would be to use "oracles" (i.e., query an external API to the smart contract to do this work). So even if it is possible to use oracle, this solution cannot be used in this work because it would go against the main goal of this research which is to propose a decentralized solution (since the external API is a centralized element and so if used, it would be like implementing a classical PKI).

Then, there are also implementations of RSA not using the "BIG Number" library created by some users. The problem of using this code is that it is not maintained and works only with very old versions of Solidity. Then, the most important point is that these implementations work only with keys of very small size (Max 256bits in practice given that the calculation for bigger keys would be much too expensive in terms of "gas"). So this solution is only valid for a "toy example". <sup>5</sup>.

Another possible solution for this problem would be to add an access list (on the controller's Ethereum addresses) on the contract that the device manufacturer would be responsible for managing.

---

<sup>5</sup>Source: <https://github.com/axic/ethereum-rsa> (<https://github.com/ethereum/EIPs/issues/74>: thread explaining that this code has been made obsolete by the introduction of the big num library)

# Chapter 10

## Results

The purpose of this evaluation is to determine the suitability of this framework for the Internet of Things. Indeed, the IoT devices are intended to be as small and cheap as possible in order to be attractive to the consumer. Therefore in order for this framework to be used, it must be as transparent as possible for the devices.

### 10.1 Experimental setup

The Proof of Concept was developed and tested on a "Manjaro Linux x86\_64" machine with an "Intel i5-8250U (8) @ 3.400GHz" CPU and 8Gb of RAM. As a matter of facts, the IoT device did not have all the resources. The device was hosted on a Virtual Machine running "Ubuntu Core 18". Ubuntu Core is a light weight version of Ubuntu targeted to constrained devices (The key characteristics of this operating system can be seen in the Table A.1). The big advantage of such a setup is that we can play with the VM resources. For testing, we decided to allocate one CPU Hyper thread<sup>1</sup> and 1Gb of RAM to the VM. We choose these numbers in order to be as close as possible to a standard Raspberry pie<sup>2</sup>. The controller used the rest of the resources (i.e., 7 hyper threads and 7Gb of RAM).

### 10.2 Device benchmarks

All the data used in the following benchmarks was gathered using the "sar" command. The data points were taken at the same time and at the same time interval (each second). Therefore, these benchmarks constitute an overall picture of the device while running the "register" phase of the Proof of Concept. It should be noted that the only processes that were running on the device at the monitoring time were the proof of concept, the SSH session (used to interact with the device), and some unrelated background tasks. (If needed, there are additional benchmarks in Section B.)

---

<sup>1</sup>Hyper-Threading is Intel's term for simultaneous multithreading (SMT). This is a process where a CPU splits each of its physical cores into virtual cores, which are known as hyper threads.

<sup>2</sup><https://www.raspberrypi.org/>

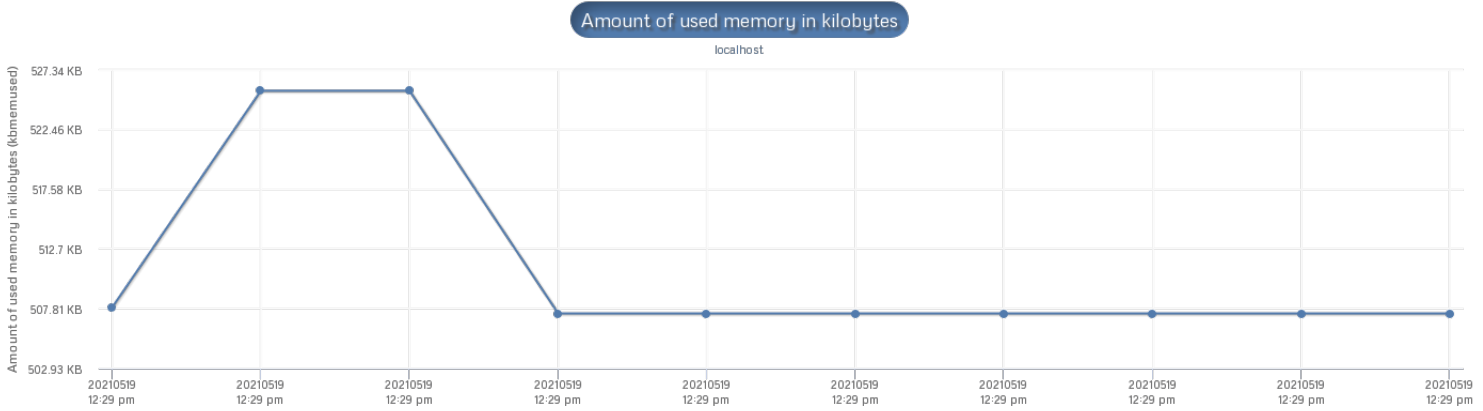


Figure 10.1: Amount of used memory in kilobytes. (This does not take into account memory used by the kernel itself.)

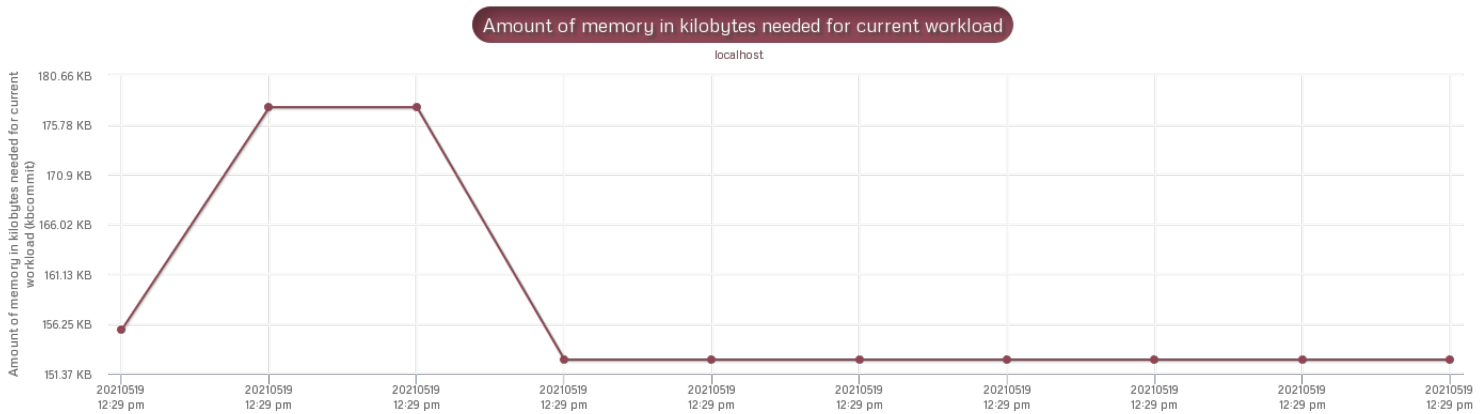


Figure 10.2: Amount of memory in kilobytes needed for current workload. (This is an estimate of how much RAM/swap is needed to guarantee that there never is out of memory.)

### 10.2.1 Memory usage

The Figure 10.1 depicts the amount of memory used by the device. Due to the fact that the memory available (1Gb) is huge compared to the peak memory usage (530Kb for the user and +- 100Mb for the OS), we consider that this Figure represents the maximal memory usage of the framework (+ the SSH session used to launch the program). On the other hand, an estimation of the minimal memory usage can be seen on the Figure 10.2. As we know that these Figures depict the whole memory usage of the user, we decided to also monitor it after the PoC was done running (i.e., after the fourth data point). This was made in order to subtract the memory usage from other processes and therefore find the memory usage of the framework. The result is that the memory footprint of the framework is about 20Kb in both cases (i.e.,  $527\text{Kb} - 507\text{Kb} = 20\text{Kb}$  and  $178\text{Kb} - 154\text{Kb} = 24\text{Kb}$ ).

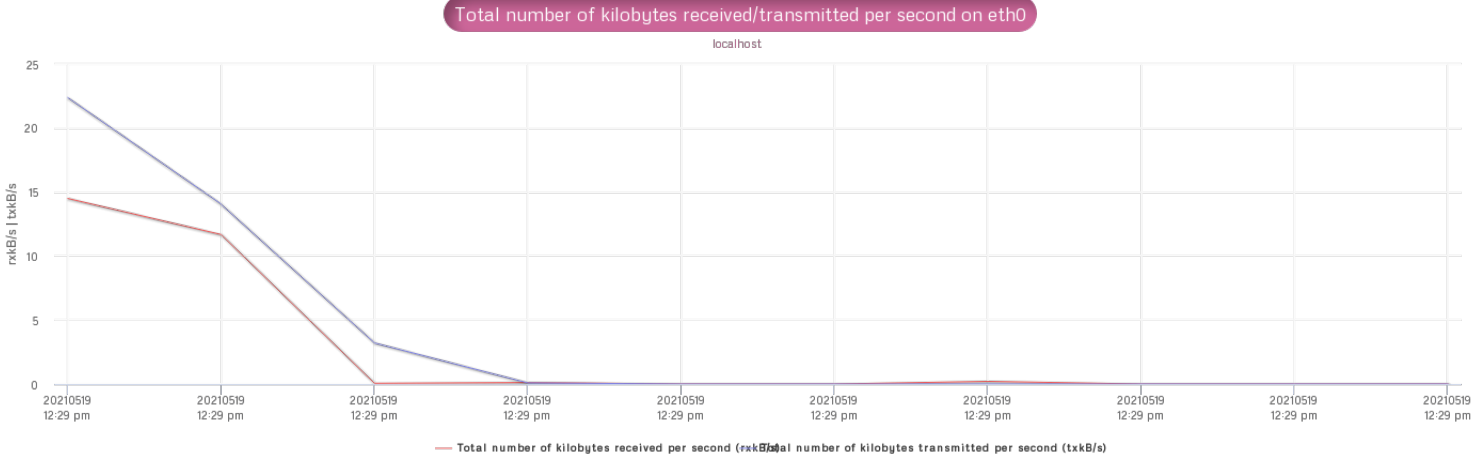


Figure 10.3: Amount of Kb sent and received over the network.

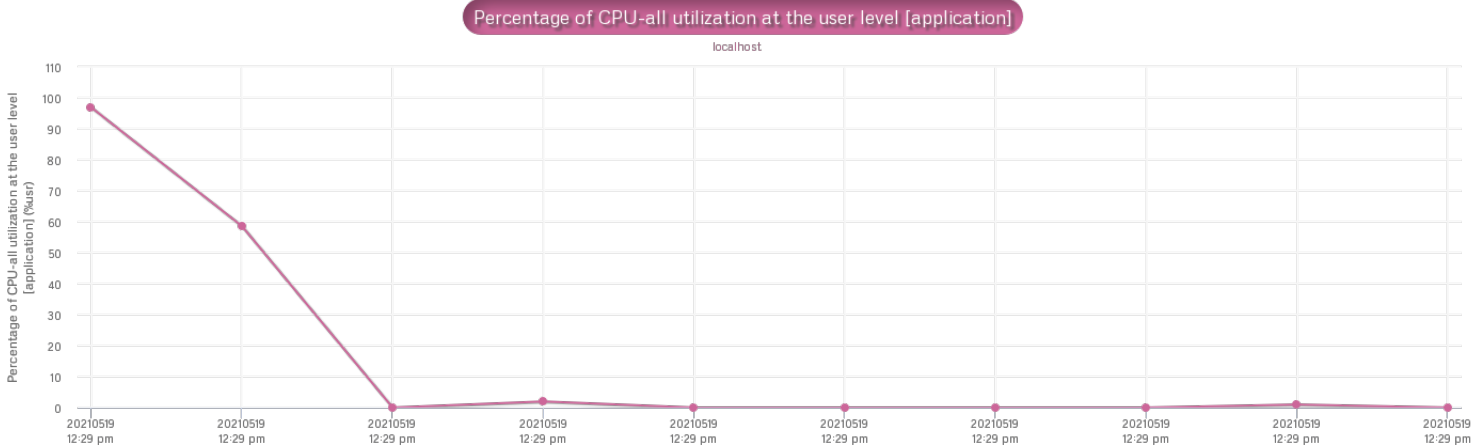


Figure 10.4: Percentage of CPU usage by user processes.

### 10.2.2 Network usage

On the Figure 10.3 we can see the total number of kilobytes received (i.e., rxkB/s) per second and the total number of kilobytes transmitted per second (i.e., txkB/s). On this graph, we see that the other user processes are not sending or receiving over the network. Therefore, we can conclude that the framework sends at most 22Kb/s and receives at most 15Kb/s.

### 10.2.3 CPU usage

The Figure 10.4 shows the CPU usage while running the framework. The CPU peak at the first data point is mainly due to the cryptographic operations performed to create the certificate coupled with the page faults (due to the loading of the certificate in RAM). The interrupts burst and the page faults can be seen on Figure 10.6 and 10.5.

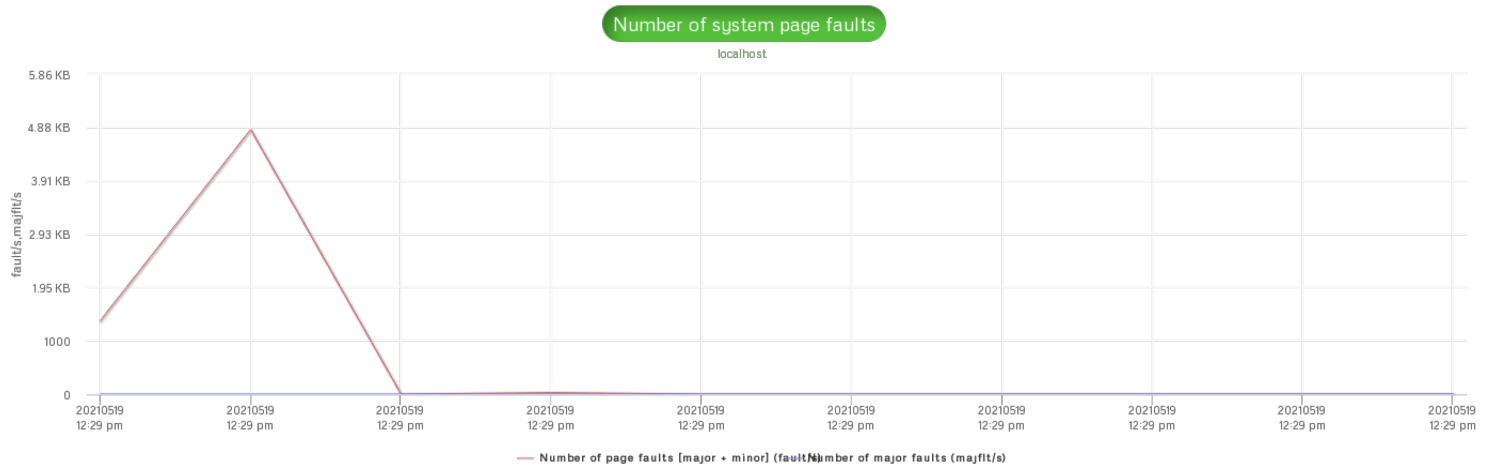


Figure 10.5: Number of system page fault.

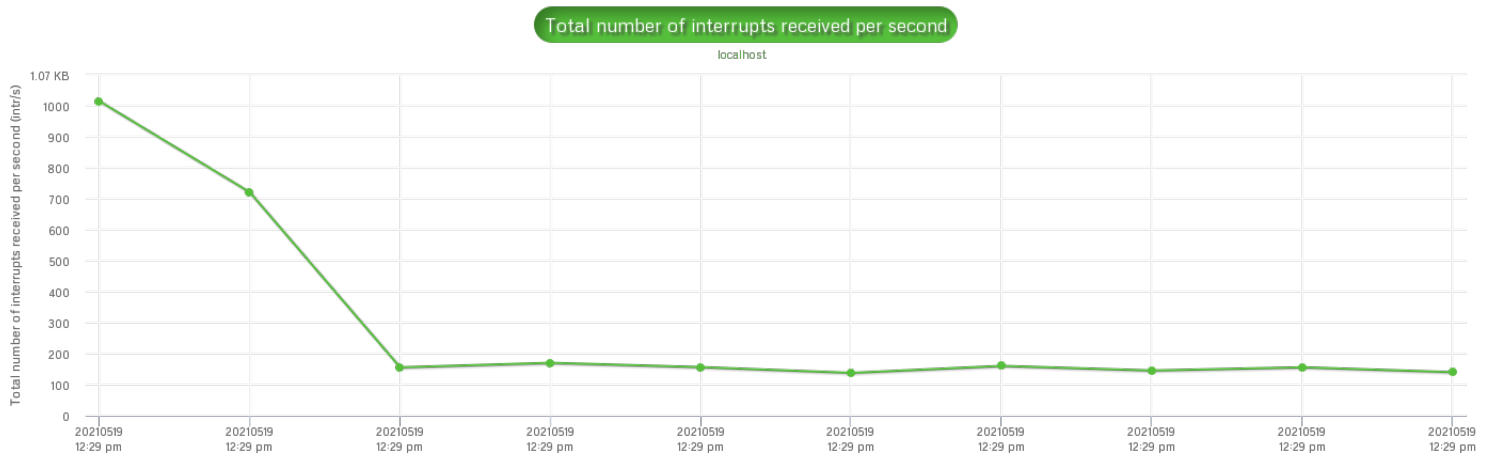


Figure 10.6: Number of interrupts.

#### **10.2.4 Disk usage**

The disk usage of the framework is about 6Mb. This number can be further reduced by using a more lightweight certificate format like XIOT[20].

**Part V**

**Conclusion**

# Chapter 11

## Discussion

### 11.1 Comparison of the different solutions

In this section, we will compare the proposed framework with the State of the Art (SOTA) which is exposed in Section 3 and we will especially focus on PKI4IOT [20] due to the purpose of this thesis (i.e., propose a decentralized alternative to the C-PKI).

Firstly, the major difference between our framework and PKI4IOT is that our framework is decentralized and thus does not rely on centralized entity like the Certificate Authority. The main benefit of our solution is that trust relies on the framework itself rather than on a private entity (i.e., the CA) which serves as corruptible central point of vulnerability. On top of that, a decentralized framework offers auditability which permits to each party to verify transactions. It is also possible to obtain such a feature with C-PKI, for example there is the log-based PKI from Google[27]. However, unlike for our framework it is an additional feature that adds weight on the IoT device.

Furthermore, we are going to discuss one by one the different drawbacks of C-PKI (exposed in Section 1.1) and we will compare with our framework in these different cases:

	<b>Our framework</b>	<b>PKI4IOT<sup>1</sup></b>
<b>Certificate signing cost:</b>	$\sim 0.28\$^2$	100-1000\$
<b>Certificate signing process time:</b>	$\sim 1.2\text{min}^3$	few days

Table 11.1: Comparison of our framework with PKI4IOT

For information, the signing cost has been obtained by taking the average cost of the transactions on the deployed contract <sup>4</sup> multiplied by the current Ether price and the processing time is the average time needed to validate a transaction in Ethereum.

<sup>1</sup>Due to their point in their Section 5.4 explaining that current certificate prices and certificate delivery time are unacceptable. We assumed that their current solution followed the CA pricing range and delivery range exposed in Section 1.1

<sup>2</sup>Assuming a price of the Ether of 2500\$ and a gas price of 3 Gwei (0.000000003 Ether)

<sup>3</sup> $6 \times 13\text{sec} = 78\text{sec}$  (6 blocks to confirm a transaction times the median time to mine a block. Source: <https://etherscan.io/chart/blocktime>)

<sup>4</sup><https://kovan.etherscan.io/address/0x2dde419f1369a4ef6b6408245479589cab4ff974>



Nevertheless, PKI4IOT due to its centralized nature could scale far more easily as explained in Section 6 and PKI4IOT has the advantage of being more lightweight than our solution. However, our solution impact on the resources can be further reduced by implementing some of the improvements<sup>5</sup> listed in Section 9.3.

Compared to Name-Value Storage (NVS) based solutions, our solution has the advantage of being able to evolve far more easily. Indeed, the smart contract paradigm offers the possibility to add a large variety of features (due to its Turing completeness) when an NVS solution is difficult to modify. Otherwise the two solutions are similar in terms of perks.

## 11.2 Research questions

The results and the solutions comparison hereabove enables us to answer the original research questions.

**Is a blockchain-based PKI a viable alternative to the widely spread CA based PKI in the context of the Internet of Things?** The numbers shown in the results section are easily achievable by a wide variety of devices. Furthermore, our solution managed to circumvent the C-PKI shortcomings. However, our solution brings the scalability problem to the table and the proposed framework needs to be further analyzed in order to see if it meets the IoT security requirements. Nevertheless, we are confident that this solution is a strong path toward a viable alternative to the CA based PKI.

**What can a blockchain-based PKI offer more than a traditional PKI?** The answer to that question is that a blockchain based PKI offers auditability natively and essentially removes the need for a corruptible central point (i.e., CA) to enable trust. On top of that, a blockchain solution is far cheaper and faster as shown in table 11.1.

## 11.3 Future works

Future works should firstly focus on enabling a path toward standardization. On top of that, future works should port this solution to embedded operating systems like zephyr<sup>6</sup>, contiki<sup>7</sup> and many more. Finally, future research should aim at including the improvements, listed in Section 5.9, into the framework.

---

<sup>5</sup>i.e., using COAP instead of HTTP and switching to XIOT instead of X.509 certificates

<sup>6</sup><https://docs.zephyrproject.org/latest/>

<sup>7</sup><https://www.contiki-ng.org/>

## Chapter 12

# Final conclusion

This thesis purpose was to put forward a viable alternative to the Public Key Infrastructure based on Certificate Authority in the context of the Internet of Things. This research is motivated by the fact that certificates have been distributed to the wrong people in the past by some CAs for questionable reasons. Indeed, the CAs represent a corruptible central point that this work wants to remove. We started by examining the different solutions available and we created a framework that adapts the state of the art PKI to a decentralized solution. Then, we designed and implemented a proof of concept in order to adequately test our solution.

We found through the results that the solution was adequate for the Internet of Things while solving the shortcomings of the standard C-PKI.

Further research should aim at standardizing the solution while porting it to embedded operating systems. However, future works should also take into consideration the scalability issue of the blockchain.

We are convinced that this work is a way to make the CAs obsolete. We hope that this work will drive research toward a new PKI paradigm.

# Bibliography

- [1] "D. Harkins" "M. Pritikin" "P. Yee". *Enrollment over Secure Transport*. RFC 7030. IETF, Oct. 2013. URL: <https://tools.ietf.org/html/rfc7030>.
- [2] Amazon. 'Amazon web services iot'. In: (2017).
- [3] Elaine Barker et al. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2020.
- [4] Mustafa Al-Bassam. 'SCPki: A smart contract-based PKI and identity system'. In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. 2017, pp. 35–40.
- [5] Stefan Beyer. *Proof-of-Work is not a Consensus Protocol: Understanding the Basics of Blockchain Consensus*. Apr. 2019. URL: <https://medium.com/>.
- [6] Francis Boily. *Explaining Ethereum Test Networks And All Their Differences*. May 2018. URL: <https://medium.com/>.
- [7] C. Bormann and Z. Shelby. *Block-wise transfers in CoAP draft-ietf-core-block-17*. Mar. 2015. URL: <https://tools.ietf.org/html/draft-ietf-core-block-17>.
- [8] Kenneth Choi. *Introduction to PKI (Public Key Infrastructure)*. May 2018. URL: <https://medium.com/>.
- [9] Comodo. 'Comodo fraud incident'. In: (March 2011).
- [10] Vincent Durham. *NAMECOIN*. 2010.
- [11] Carl Ellison and Bruce Schneier. 'Ten risks of PKI: What you're not being told about public key infrastructure'. In: *Comput Secur J* 16.1 (2000), pp. 1–7.
- [12] Emercoin. 2009. URL: <https://emercoin.com>.
- [13] CANONICAL UBUNTU ENGINEERING and SERVICES. *Ubuntu Core - Security Whitepaper*. Nov. 2018.
- [14] Nelly Fazio and Antonio Nicolosi. 'Cryptographic accumulators: Definitions, constructions and applications'. In: *Paper written for course at New York University: www.cs.nyu.edu/nicolosi/papers/accumulators.pdf* (2002).

- [15] Mohamed Amine Ferrag et al. 'Blockchain Technologies for the Internet of Things: Research Issues and Challenges'. In: *IEEE Internet of Things Journal* 6.2 (Apr. 2019), pp. 2188–2204. ISSN: 2372-2541. DOI: 10.1109/jiot.2018.2882794. URL: <http://dx.doi.org/10.1109/JIOT.2018.2882794>.
- [16] Hard Fork. 'These are the top 10 programming languages in blockchain'. In: (2019). URL: <https://thenextweb.com/hardfork/2019/05/24/javascript-programming-java-cryptocurrency/>.
- [17] Filip Forsby et al. 'Lightweight X.509 Digital Certificates for the Internet of Things: Third International Conference, InterIoT 2017, and Fourth International Conference, SaSeIoT 2017, Valencia, Spain, November 6-7, 2017, Proceedings'. In: July 2018, pp. 123–133. ISBN: 978-3-319-93796-0. DOI: 10.1007/978-3-319-93797-7\_14.
- [18] Conner Fromknecht, Dragos Velicanu and Sophia Yakoubov. 'A Decentralized Public Key Infrastructure with Identity Retention.' In: *IACR Cryptol. ePrint Arch.* 2014 (2014), p. 803.
- [19] Yunhua He et al. 'A blockchain based truthful incentive mechanism for distributed P2P applications'. In: *IEEE Access* 6 (2018), pp. 27324–27335.
- [20] Joel Höglund et al. 'PKI4IoT: Towards public key infrastructure for the Internet of Things'. In: *Computers & Security* 89 (2020), p. 101658. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2019.101658>. URL: <http://www.sciencedirect.com/science/article/pii/S0167404819302019>.
- [21] Xiaohong Huang et al. 'LNSC: A security model for electric vehicle and charging pile management based on blockchain ecosystem'. In: *IEEE Access* 6 (2018), pp. 13565–13574.
- [22] IBM. 'Device democracy'. In: *IBM Institute for Business Value* (2015). URL: <https://www.ibm.com/downloads/cas/Y5ONA8EV>.
- [23] IoT Analytics. 'IoT Security Market Report 2017-2022'. In: (2017).
- [24] Desiree Johnson. *How Much Does a SSL Certificate Cost?* Dec. 2019.
- [25] Burt Kaliski. *PKCS #7: Cryptographic Message Syntax Version 1.5*. RFC 2315. <http://www.rfc-editor.org/rfc/rfc2315.txt>. RFC Editor, Mar. 1998. URL: <http://www.rfc-editor.org/rfc/rfc2315.txt>.
- [26] Loren M Kohnfelder. 'Towards a practical public-key cryptosystem.' PhD thesis. Massachusetts Institute of Technology, 1978.
- [27] B. Laurie, A. Langley and E. Kasper. *Certificate Transparency*. RFC 6962. RFC Editor, June 2013.
- [28] Gaoqi Liang et al. 'Distributed blockchain-based data protection framework for modern power systems against cyber attacks'. In: *IEEE Transactions on Smart Grid* 10.3 (2018), pp. 3162–3173.
- [29] Chao Lin et al. 'BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0'. In: *Journal of Network and Computer Applications* 116 (2018), pp. 42–52.

- [30] J. Martins et al. 'Fostering Customer Bargaining and E-Procurement Through a Decentralised Marketplace on the Blockchain'. In: *IEEE Transactions on Engineering Management* (2020), pp. 1–15. DOI: 10.1109/TEM.2020.3021242.
- [31] Microsoft. 'Azure iot suite'. In: (2017).
- [32] Microsoft. 'Erroneous VeriSign-issued digital certificates pose spoofing hazard.' In: (March 2001).
- [33] P. Morrissey, N. P. Smart and B. Warinschi. 'A Modular Security Analysis of the TLS Handshake Protocol'. In: *Advances in Cryptology - ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 55–73. ISBN: 978-3-540-89255-7.
- [34] Satoshi Nakamoto and A Bitcoin. 'A peer-to-peer electronic cash system'. In: *Bitcoin*.—URL: <https://bitcoin.org/bitcoin.pdf> 4 (2008).
- [35] M. Nystrom and B. Kaliski. *PKCS #10: Certification Request Syntax Specification Version 1.7*. RFC 2986. RFC Editor, Nov. 2000.
- [36] Pim Otte, Martijn de Vos and Johan Pouwelse. 'TrustChain: A Sybil-resistant scalable blockchain'. In: *Future Generation Computer Systems* 107 (2020), pp. 770–780.
- [37] P. Leach, M. Mealling, R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. IETF, July 2005, pp. 1–30. URL: <https://tools.ietf.org/html/rfc4122>.
- [38] Andrey Petrov. *An economic incentive for running Ethereum full nodes*. May 2018. URL: [https://medium.com/vipnode/an-economic-incentive-for-running-ethereum-full-nodes-ecc0c9ebe22#:~:text=A%20full%20node%20has%20a,memory%20\(some%20actual%20stats\).&text=A%20light%20node%20has%20only,while%20talking%20to%20full%20nodes..](https://medium.com/vipnode/an-economic-incentive-for-running-ethereum-full-nodes-ecc0c9ebe22#:~:text=A%20full%20node%20has%20a,memory%20(some%20actual%20stats).&text=A%20light%20node%20has%20only,while%20talking%20to%20full%20nodes..)
- [39] "H. Prins et al. 'Black Tulip: Report of the investigation into the DigiNotar certificate authority breach.' In: (August 2012).
- [40] Shahid Raza et al. 'SecureSense: End-to-end secure communication architecture for the cloud-connected Internet of Things'. In: *Future Generation Computer Systems* 77 (2017), pp. 40–51. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2017.06.008>. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X17312360>.
- [41] Jamie Redman. *Excessive Flooding in Sichuan Causes 20% Hashrate Losses for Chinese Bitcoin Miners*. Aug. 2020. URL: <https://news.bitcoin.com/excessive-flooding-in-sichuan-causes-20-hashrate-losses-for-chinese-bitcoin-miners/>.
- [42] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. IETF, Aug. 2018, pp. 24–77. URL: <https://tools.ietf.org/html/rfc8446>.
- [43] Gabriele Restuccia, Hannes Tschofenig and Emmanuel Baccelli. *Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3*. 2020. arXiv: 2011.12035 [cs.CR].

- [44] Y. Rosasooria et al. 'E-Voting on Blockchain using Solidity Language'. In: *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*. 2020, pp. 1–6. DOI: 10.1109/ICVEE50212.2020.9243267.
- [45] S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. IETF, May 2008, pp. 1–147. URL: <https://tools.ietf.org/html/rfc5280>.
- [46] Meruja Selvamanikkam. *Digital Signature Generation*. Feb. 2018. URL: <https://meruja.medium.com/digital-signature-generation-75cc63b7e1b4>.
- [47] Greg Slepak and Anya Petrova. *The DCS Theorem*. 2018. arXiv: 1801.04335 [cs.DC].
- [48] Peter van der Stok et al. *EST over secure CoAP (EST-coaps)*. Internet-Draft draft-ietf-ace-coap-est-09. <http://www.ietf.org/internet-drafts/draft-ietf-ace-coap-est-09.txt>. IETF Secretariat, Feb. 2019. URL: <http://www.ietf.org/internet-drafts/draft-ietf-ace-coap-est-09.txt>.
- [49] Pei-Yih Ting, Jia-Lun Tsai and Tzong-Sun Wu. 'Signcryption method suitable for low-power IoT devices in a wireless sensor network'. In: *IEEE Systems Journal* 12.3 (2017), pp. 2385–2394.
- [50] I. Vakilinia, S. Badsha and S. Sengupta. 'Crowdfunding the Insurance of a Cyber-Product Using Blockchain'. In: *2018 9th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. 2018, pp. 964–970. DOI: 10.1109/UEMCON.2018.8796515.
- [51] Qin Wang et al. 'Preserving transaction privacy in bitcoin'. In: *Future Generation Computer Systems* 107 (2020), pp. 793–804.
- [52] J. Won et al. 'Decentralized Public Key Infrastructure for Internet-of-Things'. In: *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 2018, pp. 907–913. DOI: 10.1109/MILCOM.2018.8599710.
- [53] Gavin Wood et al. 'Ethereum: A secure decentralised generalised transaction ledger'. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.
- [54] Wei Yin et al. 'An anti-quantum transaction authentication approach in blockchain'. In: *IEEE Access* 6 (2018), pp. 5393–5401.

# Appendices

## Appendix A

# Ubuntu Core key characteristics

This table depicts the key characteristics of Ubuntu Core.

<b>Minimum requirements</b>	500 Mhz single core processor 256 MB RAM 512 MB Storage
<b>Container runtimes and orchestration</b>	Snapd Docker, AWS Greengrass, Azure IoT Edge Kubernetes via Microk8s LXD
<b>Application security</b>	Isolation via AppArmor and Seccomp TPM support Secure boot support Full disk encryption
<b>Updates</b>	Automatic over the air update Atomic updates Roll-backs on failure
<b>CPU support</b>	32 bit / 64 bit, x86 / ARM

Table A.1: Ubuntu Core key characteristics. Source: [13]

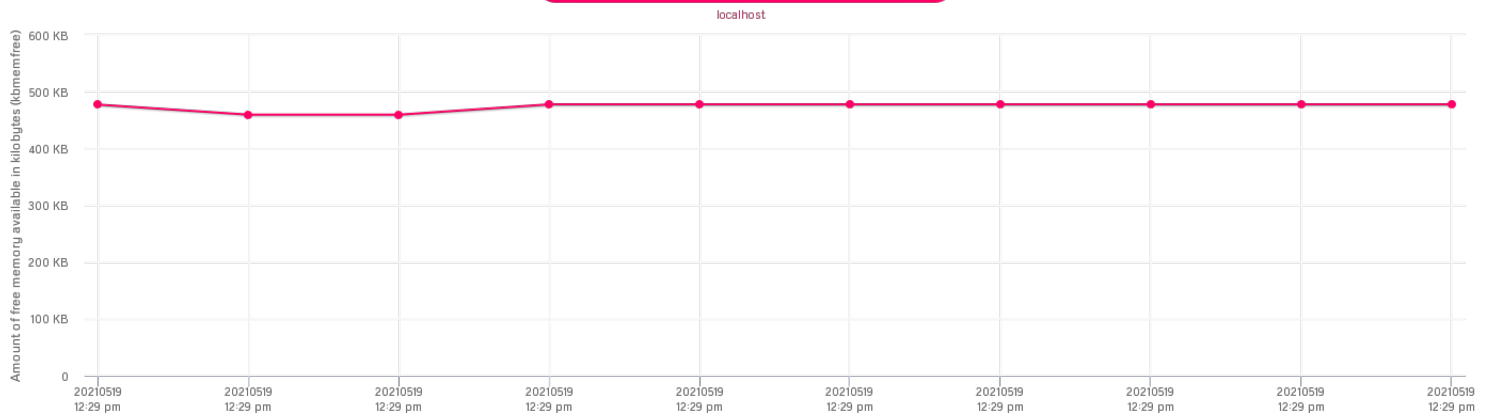


## **Appendix B**

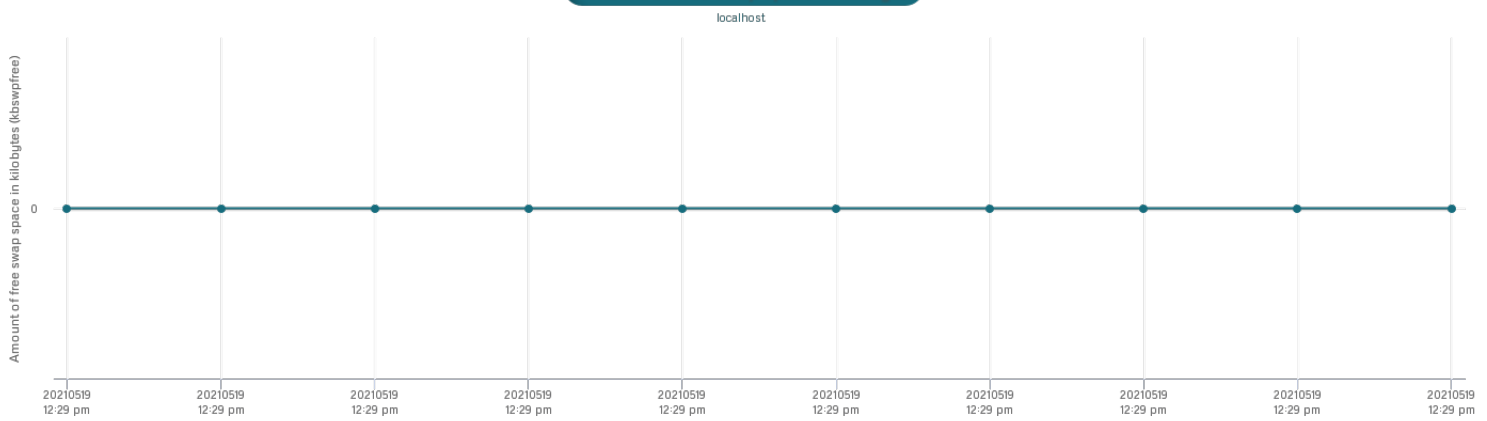
# **Additional system benchmarks of the device**

This chapter provides additional benchmarks of the device while computing the register phase. (State of the device depicted on Figures below)

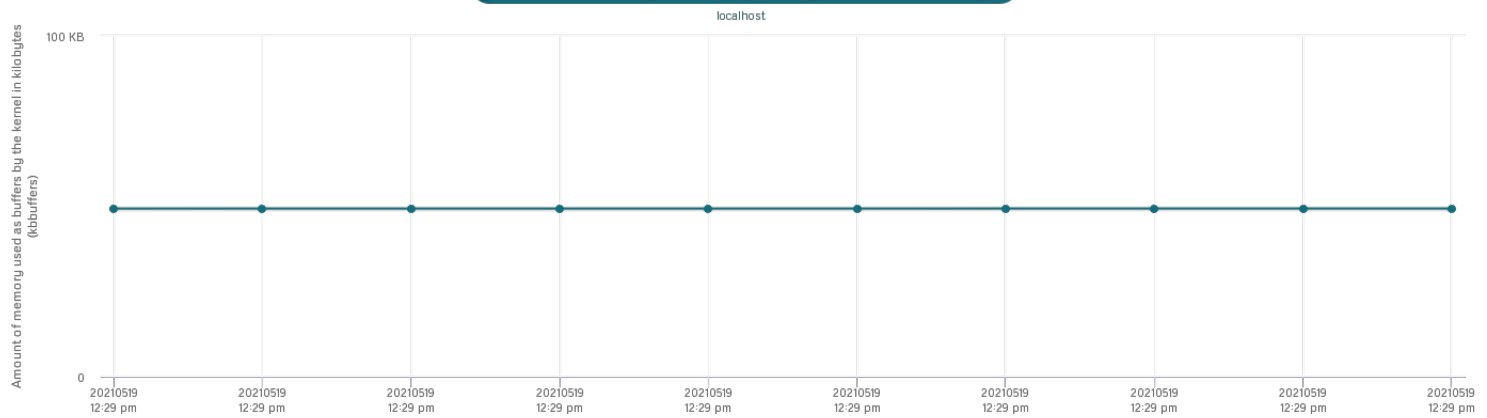
Amount of free memory available in kilobytes



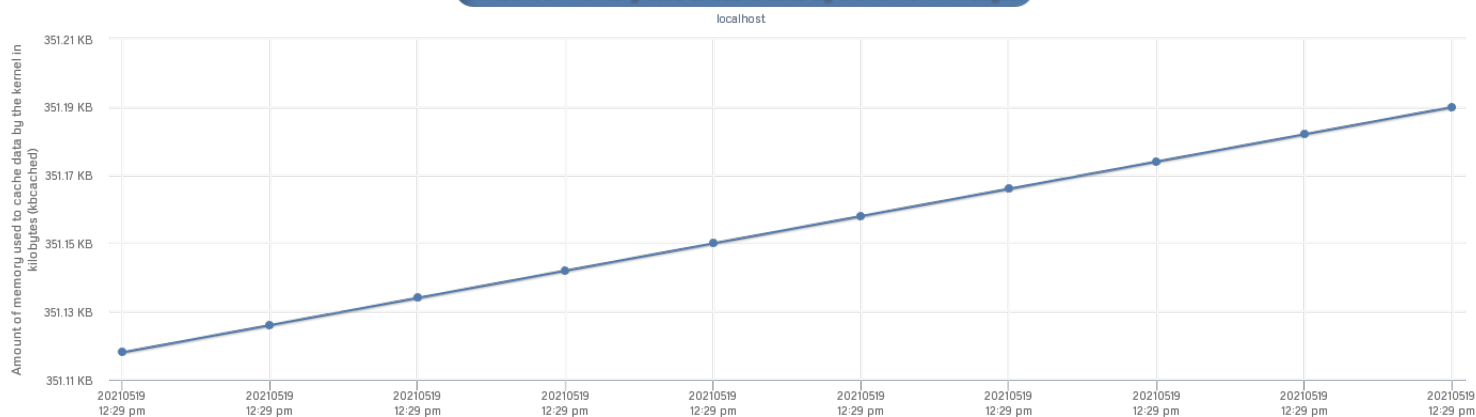
Amount of free swap space in kilobytes



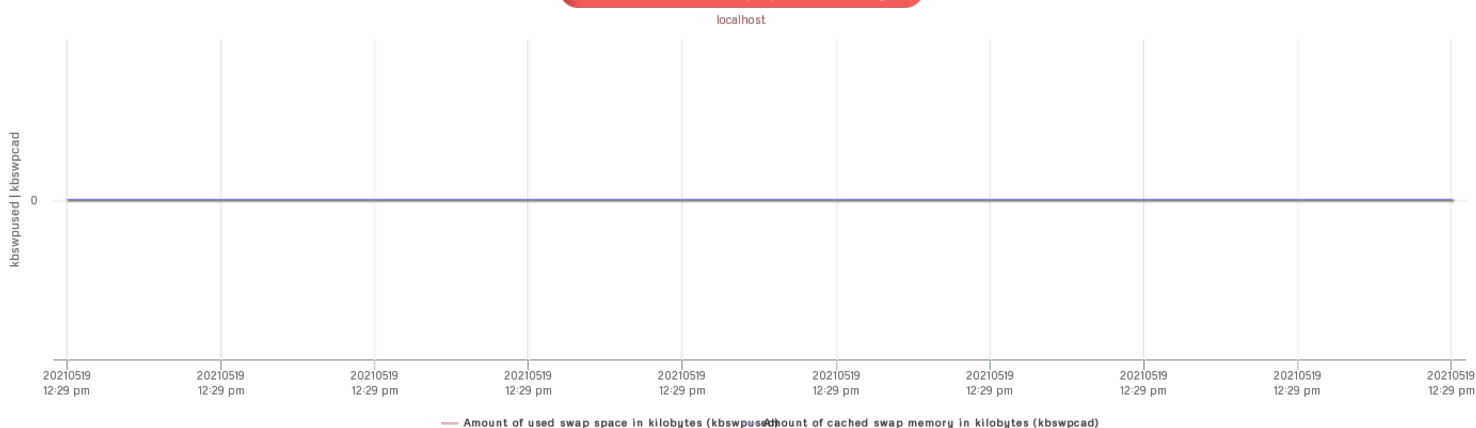
Amount of memory used as buffers by the kernel in kilobytes



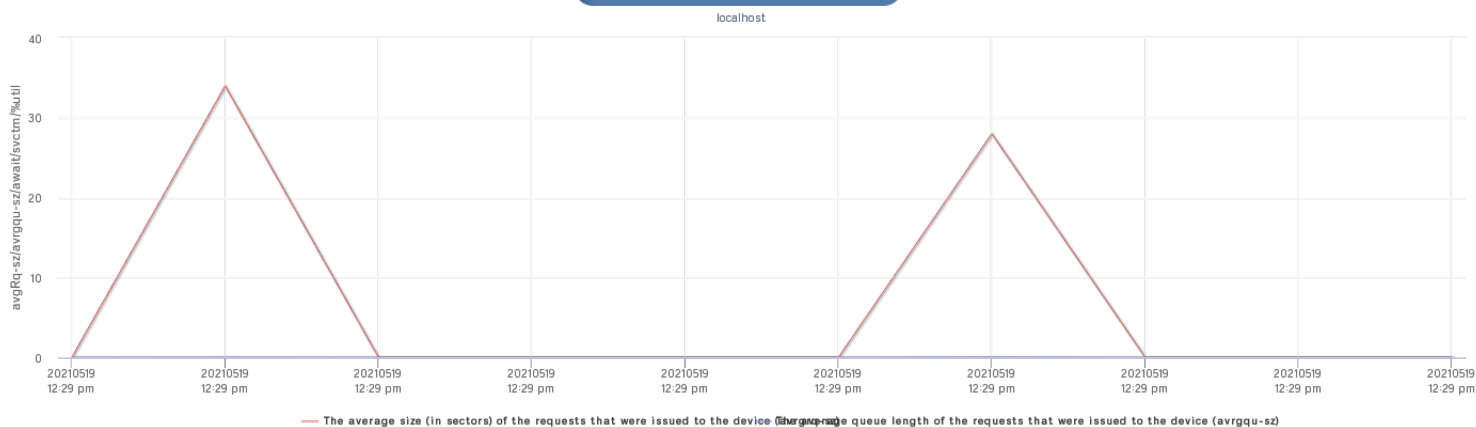
Amount of memory used to cache data by the kernel in kilobytes



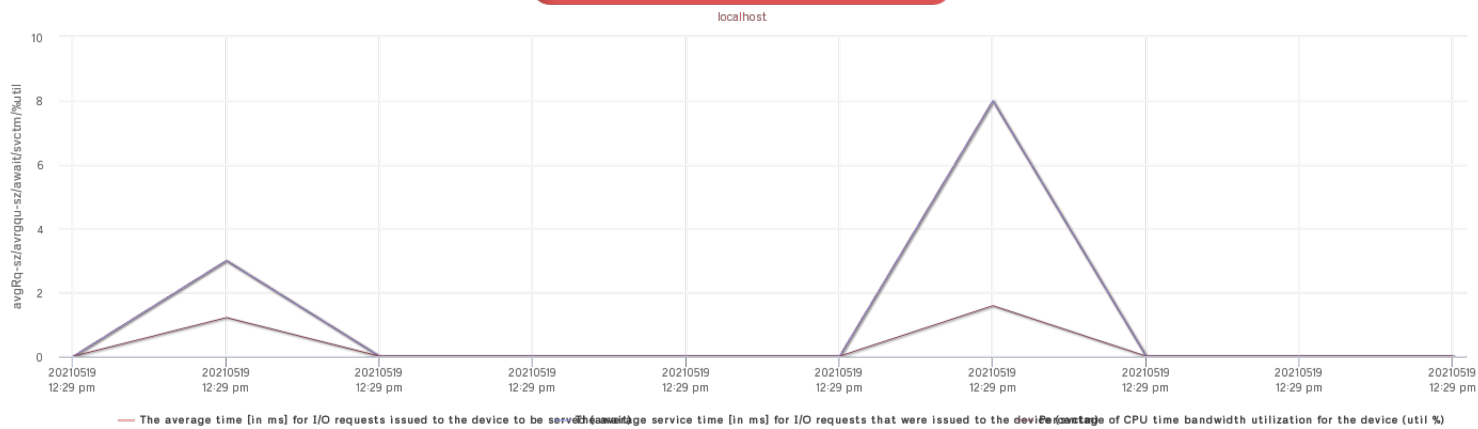
Amount of used swap space in kilobytes



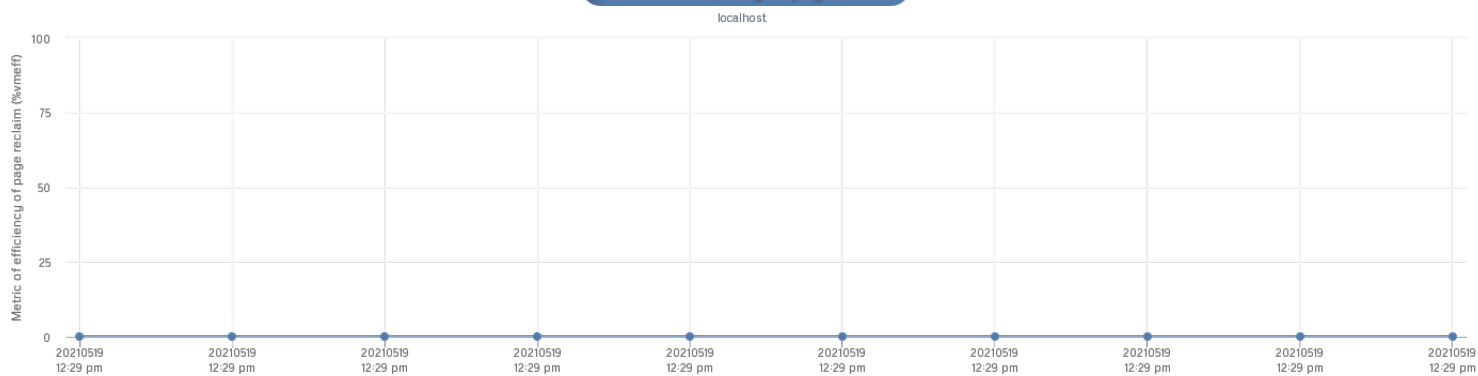
Average size/queue length to dev8-0



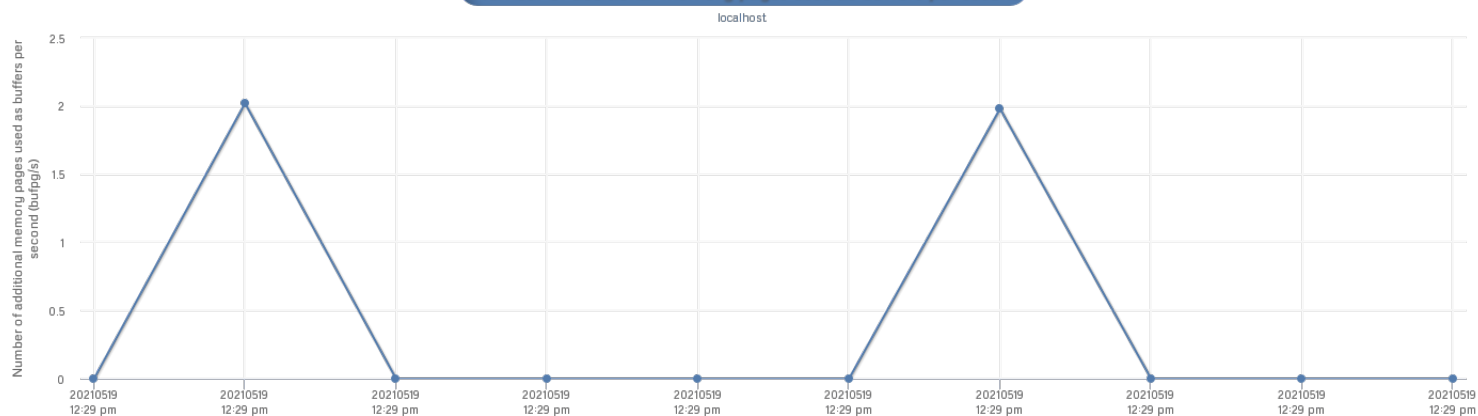
Average time/service time/utilization to dev8-0

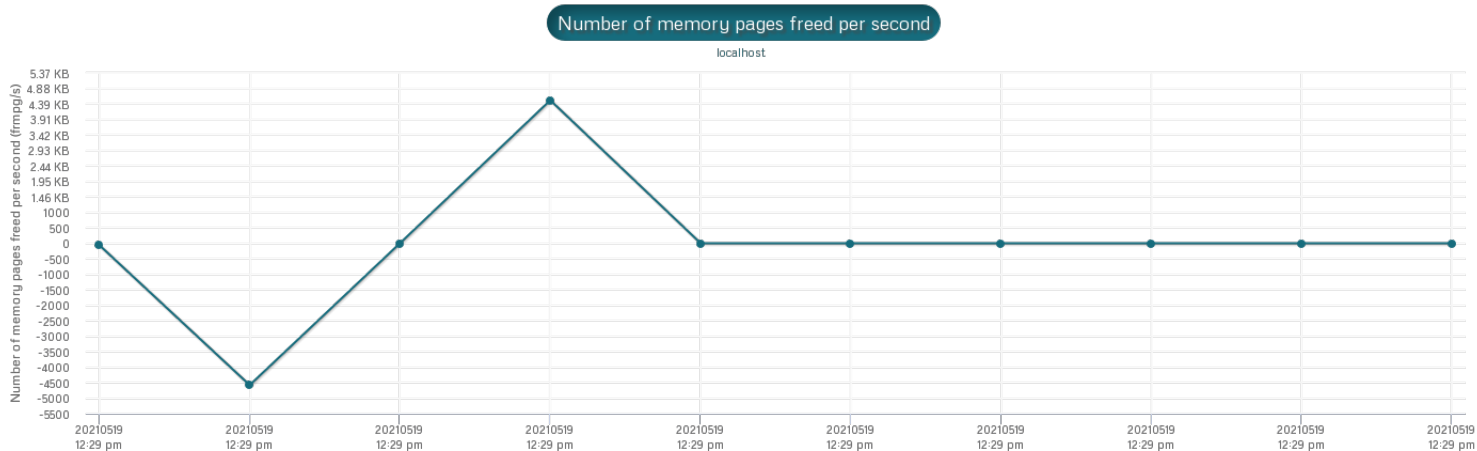
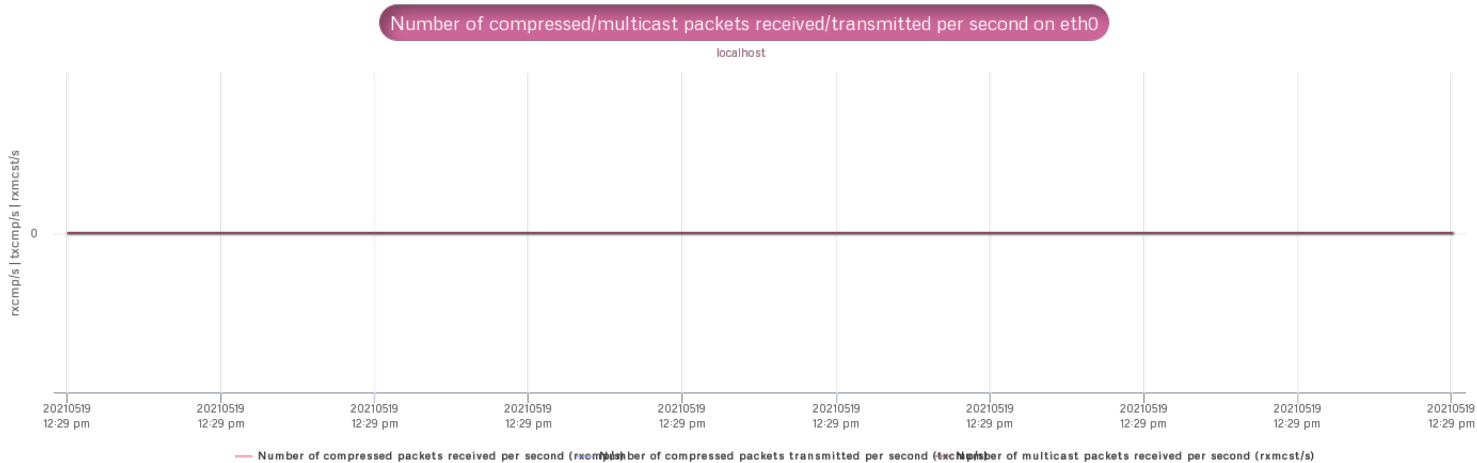
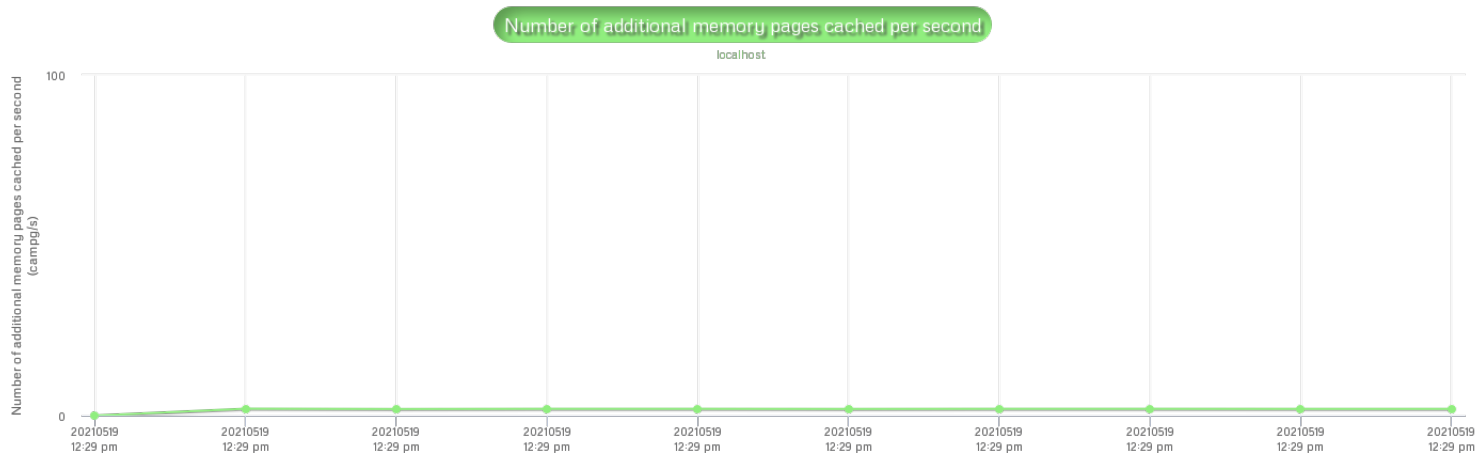


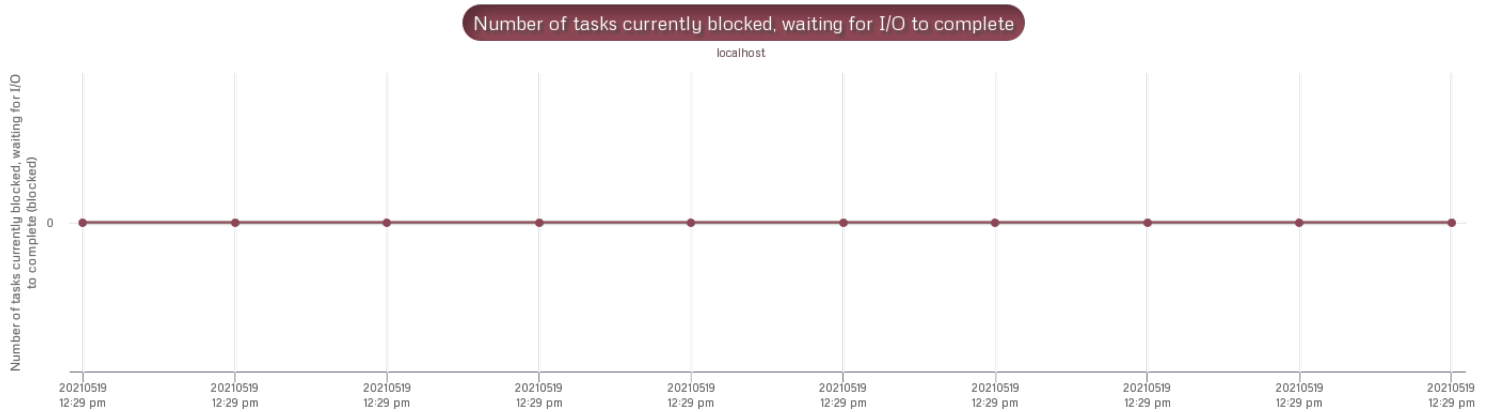
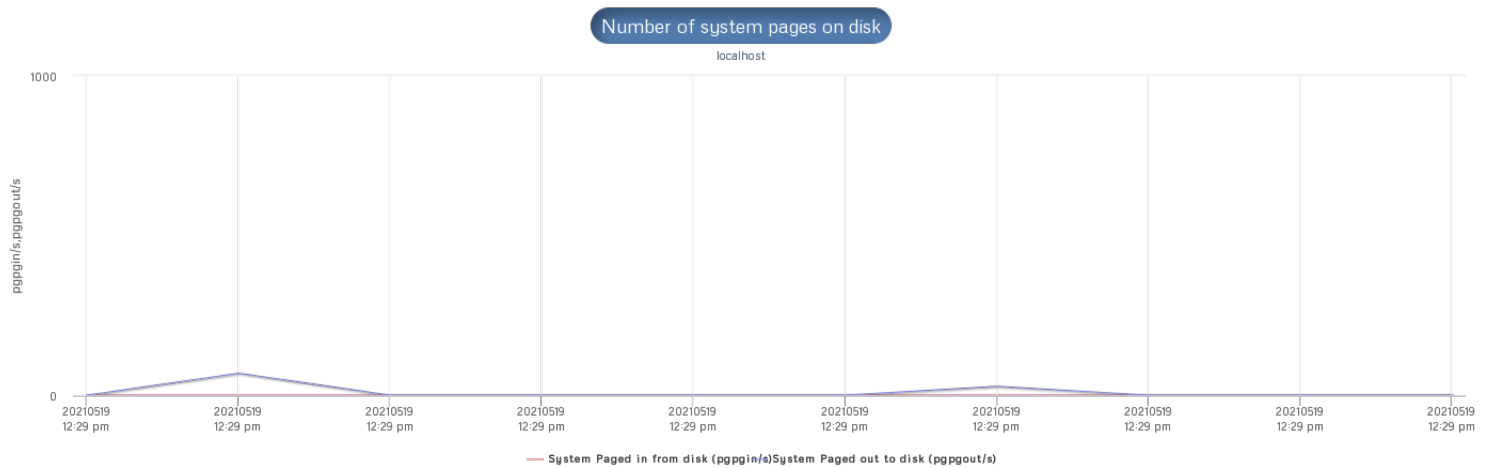
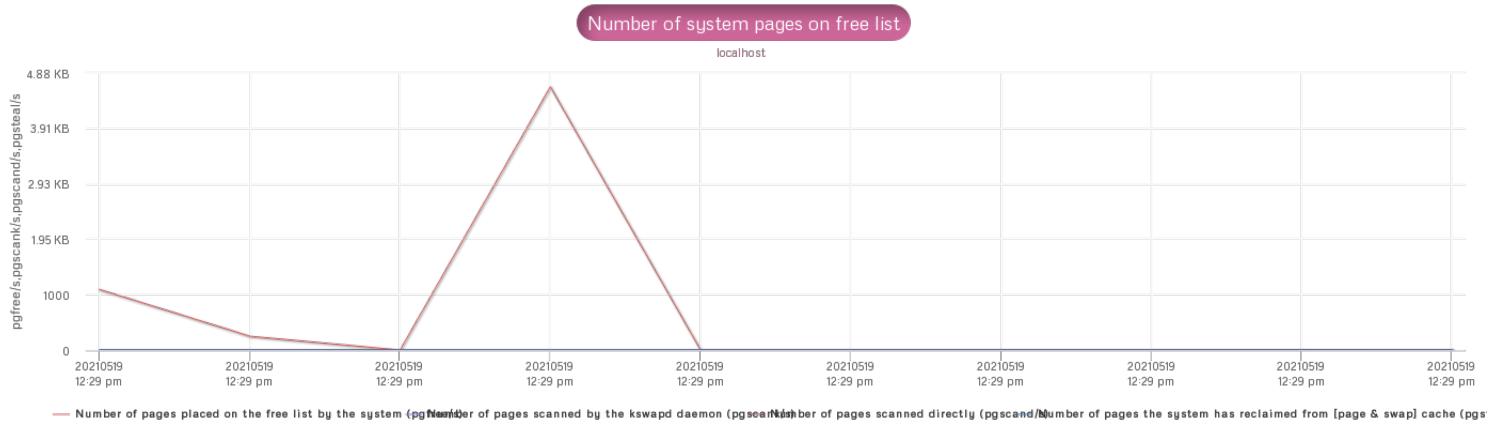
Metric of efficiency of page reclaim



Number of additional memory pages used as buffers per second

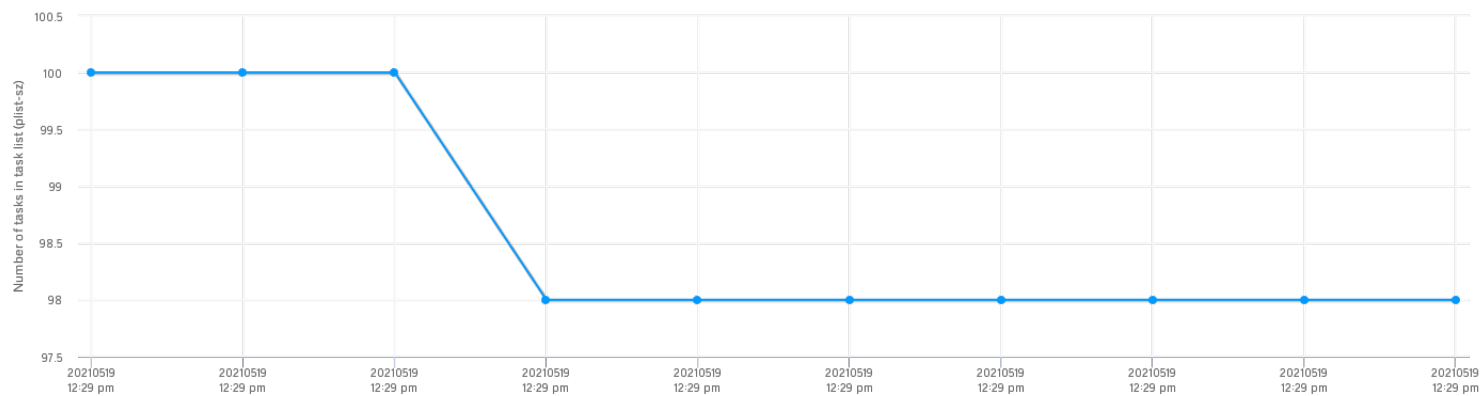






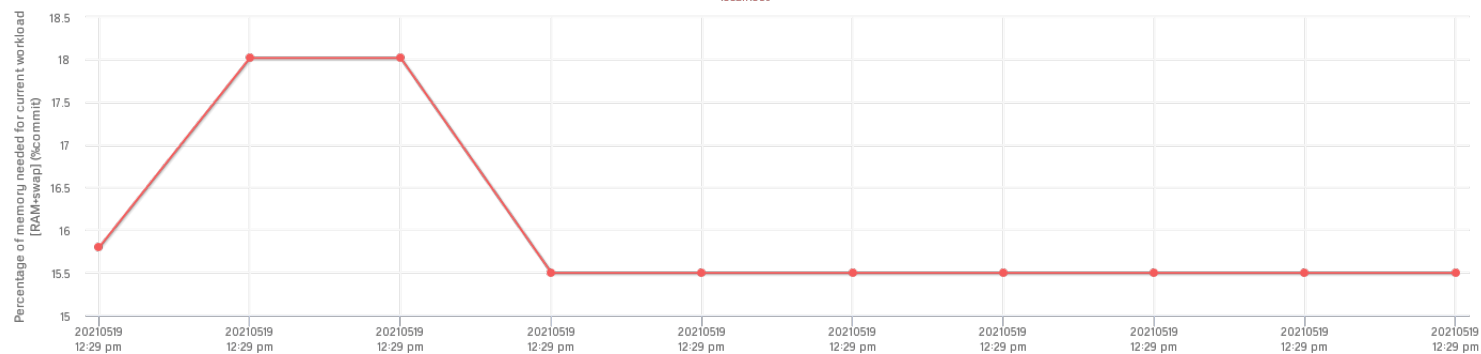
Number of tasks in task list

localhost



Percentage of memory needed for current workload [RAM+swap]

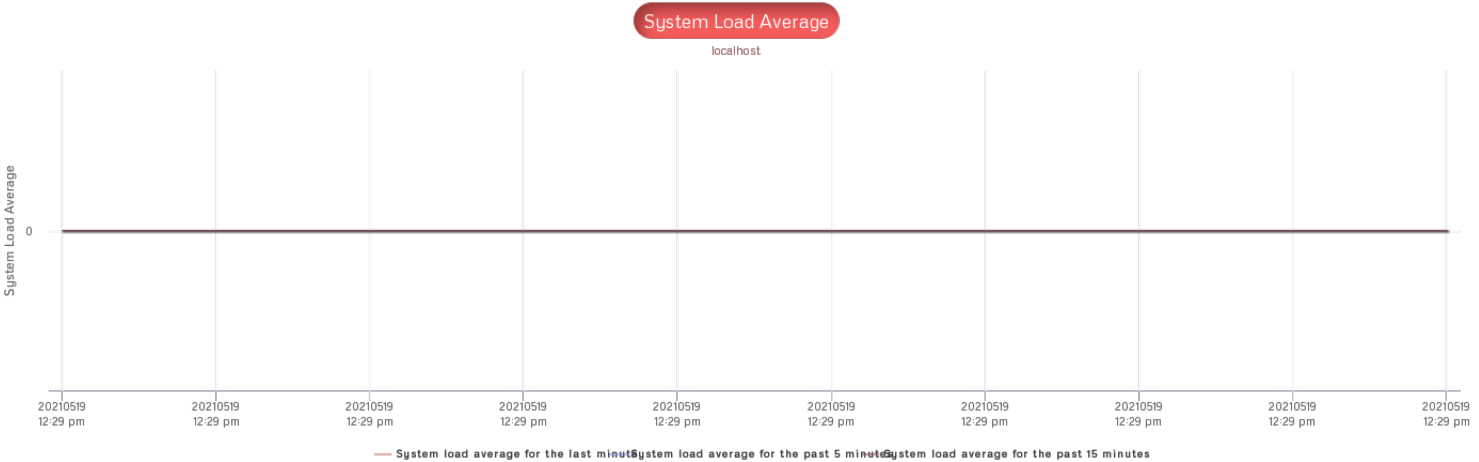
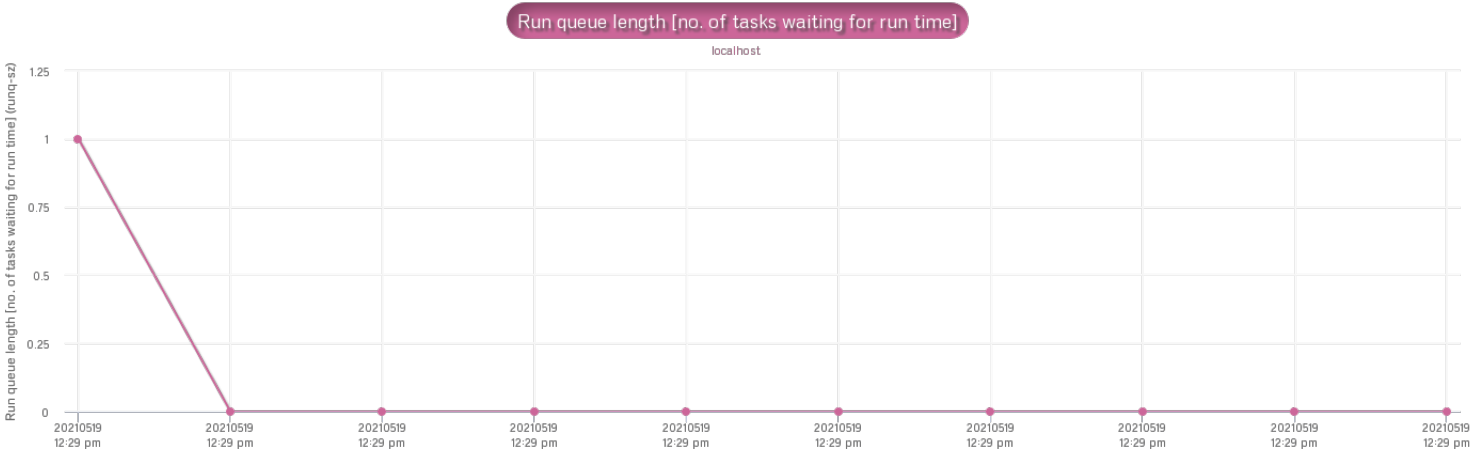
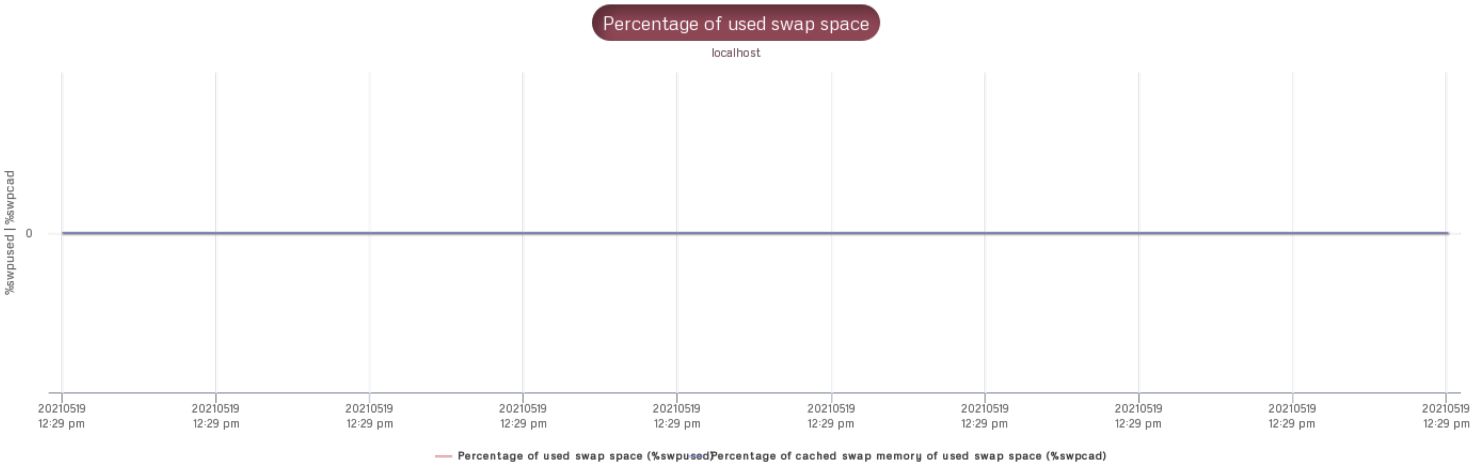
localhost



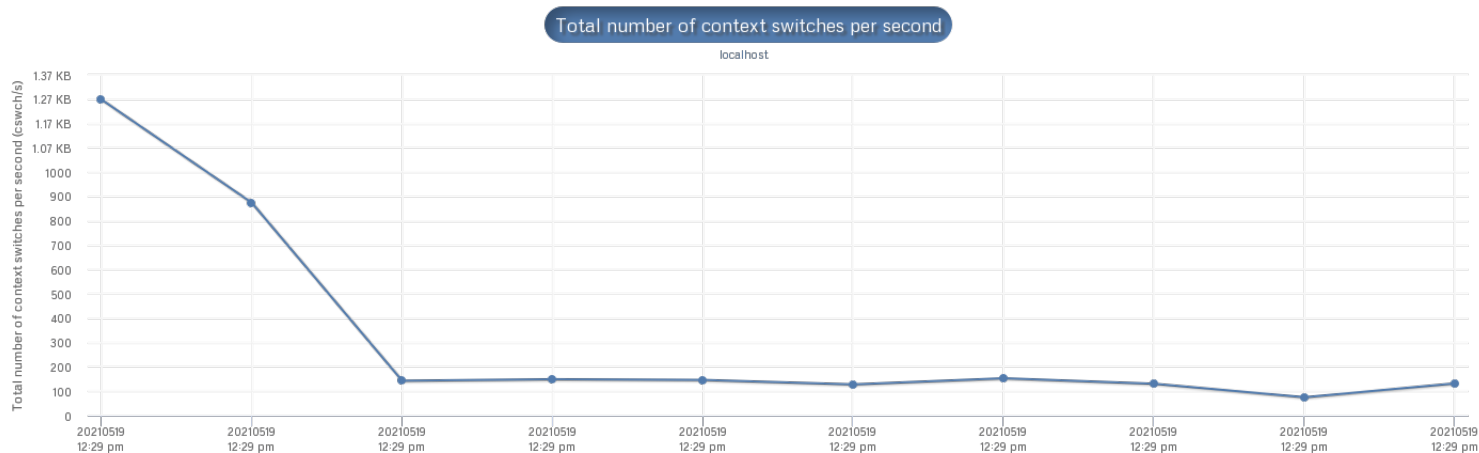
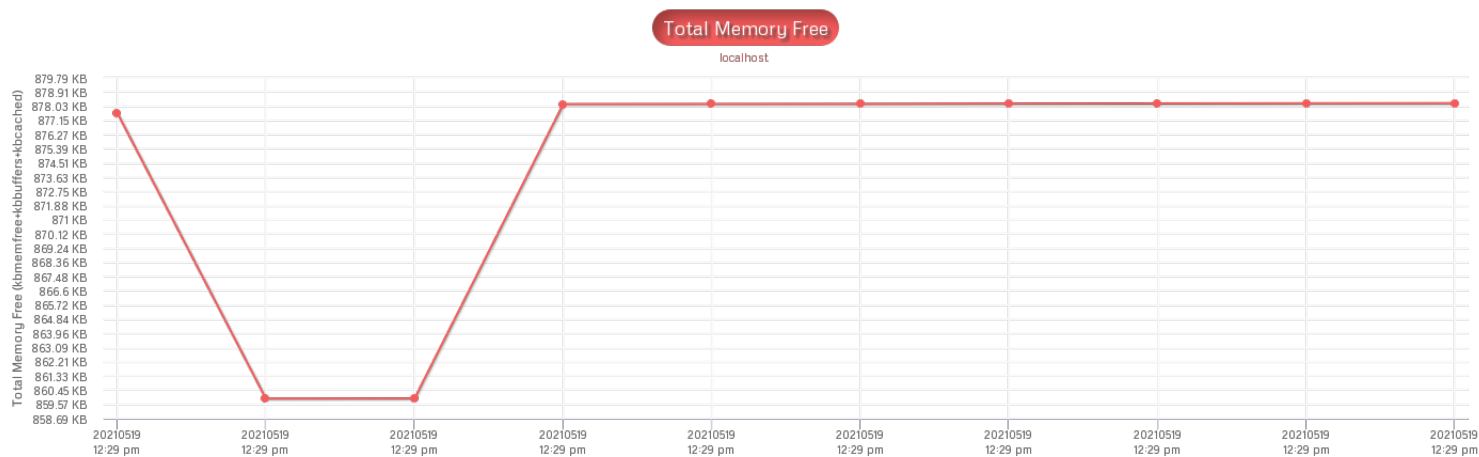
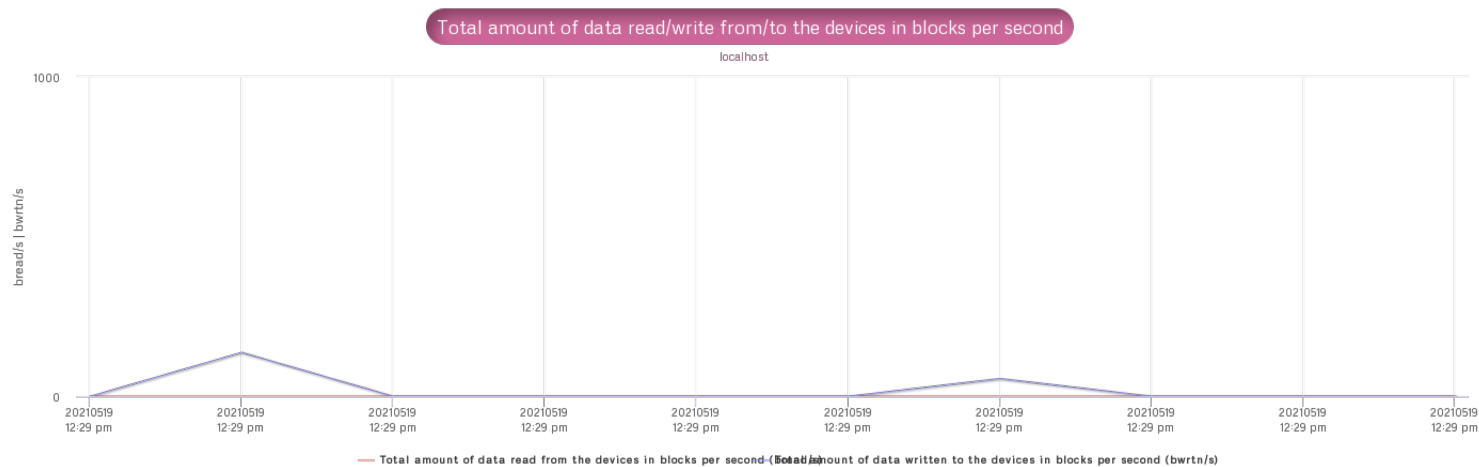
Percentage of used memory

localhost

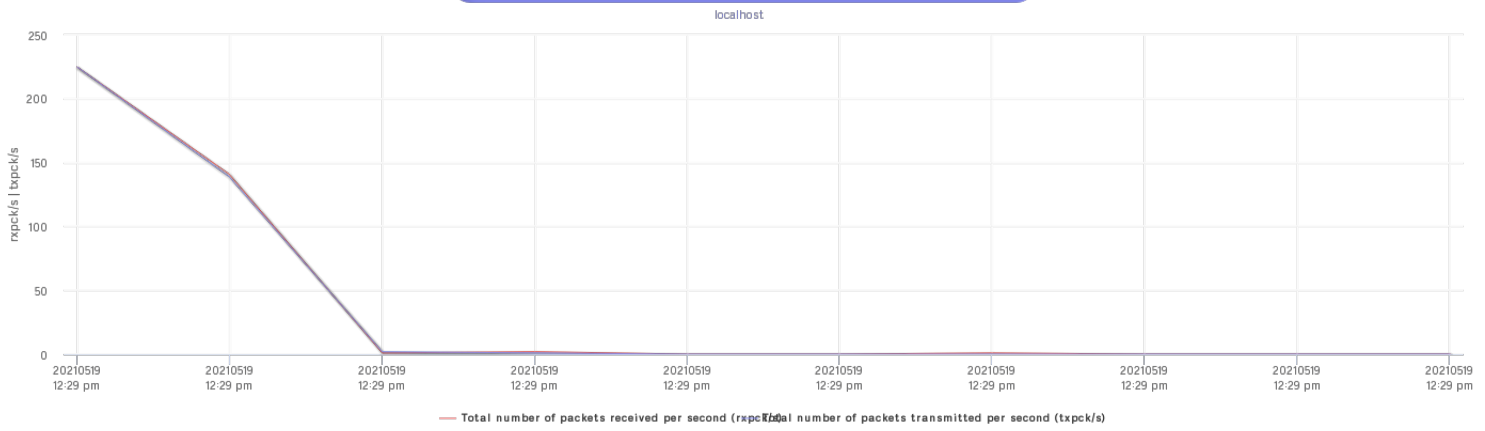




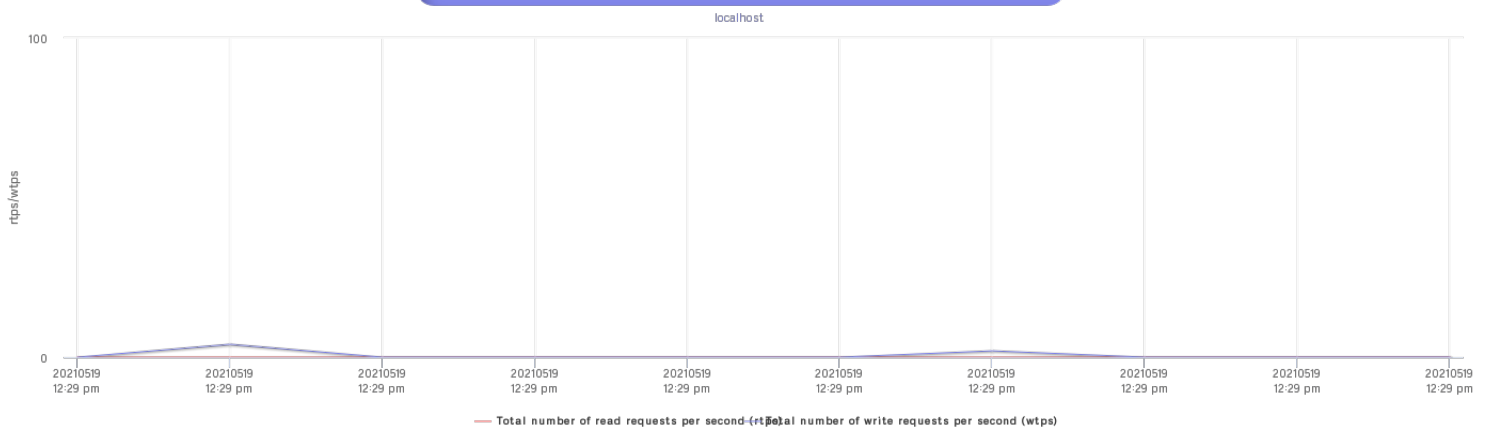




Total number of packets received/transmitted per second on eth0



Total number of read/write requests per second issued to physical devices



Total number of swap pages the system brought in per second

