Universitá degli Studi di Messina

Dottorato di Ricerca in Ingegneria Civile Ambientale e della Sicurezza

Curriculum Scienze e tecnologie, materiali, energia e sistemi complessi per il calcolo distribuito e le reti SSD INF/01

Ciclo XXXVI

How to strengthen Cloud Computing exploiting solutions to Edge

AUTHOR:
ALESSIO CATALFAMO

HEAD OF THE DOCTORAL SCHOOL:
Prof. Dr. Gaetano Bosurgi

ADVISOR:
Prof. Dr. Massimo Villari

Accademic Year 2022-2023

Abstract

The present PhD thesis deepens Cloud Computing's known issues proposing different solutions in order to overcome them. The proposed solutions in the thesis exploit Edge Computing, in some cases, in combination with Cloud Infrastructure realizing a Cloud-Edge Continuum Infrastructure. The thesis has delved into critical aspects of Cloud Computing, specifically focusing on the distribution of computation and security. In addressing these challenges, the perspectives of both user data and the computational processes applied to them were carefully considered. For Data management the realized work describes different solutions related to the use of Edge Computing combined with Homomorphic Encryption and the distribution among different entities of the Identity and Access Manager Component. For the Computation of Data, the work proposes different solutions related to the execution of complex processes in Edge Computing infrastructure. Throughout the thesis, various solutions have been proposed to effectively address the identified limitations. Furthermore, the research includes multiple experimental evaluations to validate the proposed solutions.

Keywords: Cloud Computing, Edge Computing, Cloud-Edge Continuum, Edge Intelligence

Contents

List of Figures

List of Tables

Earlier Publications

This thesis is the outcome of the doctoral degree started three years ago. It is based on selected works (listed here below) already published, under review or submitted. Parts of these papers are contained in verbatim.

[1] A. Catalfamo, A. Ruggeri, A. Celesti, M. Fazio and M. Villari, "A Microservices and Blockchain Based One Time Password (MBB-OTP) Protocol for Security-Enhanced Authentication" 2021 IEEE Symposium on Computers and Communications (ISCC), Athens, Greece, 2021, pp. 1-6

[2] A. Catalfamo, M. Fazio, F. Martella, A. Celesti and M. Villari, "MuoviMe: Secure Access to Sustainable Mobility Services in Smart City" 2021 IEEE Symposium on Computers and Communications (ISCC), Athens, Greece, 2021, pp. 1-5

[3] C. Sicari, A. Catalfamo, A. Galletta and M. Villari, "A Distributed Peer to Peer Identity and Access Management for the Osmotic Computing" 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Taormina, Italy, 2022, pp. 775-781

[4] A. Catalfamo, A. Celesti, M. Fazio and M. Villari, "A Homomorphic Encryption Service to Secure Data Processing in a Cloud/Edge Continuum Context" 2022 9th International Conference on Future Internet of Things and Cloud (FiCloud), Rome, Italy, 2022, pp. 55-61

[5] A. Catalfamo, A. Celesti, M. Fazio, G. Randazzo and M. Villari, "A Platform for Federated Learning on the Edge: a Video Analysis Use Case" 2022 IEEE Symposium on Computers and Communications (ISCC), Rhodes, Greece, 2022, pp. 1-7

[6] A. Catalfamo et al., "Scaling Data Analysis Services in an Edge-based Federated Learning Environment," 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), Vancouver, WA, USA, 2022, pp. 167-172

[7] V. Lukaj, A. Catalfamo, F. Martella, M. Fazio, M.Villari, A. Celesti, "A NoSQL DBMS Transparent Data Encryption Approach for Cloud/Edge Continuum", 2023 IEEE Symposium on Computers and Communications (ISCC), Tunis, Tunisia, 2023 (Accepted)

CHAPTER 1

Introduction

Cloud Computing was one of the most disruptive innovations introduced during the last decades. It changed the approach to any kind of computational resources introducing many advantages related to the latter's management. This transformative innovation has completely reshaped the landscape of Information and Communication service delivery, introducing a level of transparency that enables end-users to effortlessly access a multitude of resources through a simple internet connection all while leveraging the cost-effective pay-per-use model. Indeed, the introduction of Cloud Computing involves a paradigm shift that undeniably offers tremendous business advantages while simultaneously raising well-documented concerns. It allows final users to access several services only with an Internet connection making transparent every kind of configuration and maintenance of those services.

Although, as said, this new paradigm brings several advantages to final users, it's impossible to ignore that the strong abstraction provided by Cloud Computing introduces some known issues [8] that can be briefly summarized in three points:

- Security of end users' data that is hypothetically threatened by Cloud providers access [9].
- Utilization of needed bandwidth for remote service access [10]. Indeed, as specified, Cloud Services are mainly accessed through an Internet connection.
- Centralized approach that impacts real-time services' class requirements introducing

an overhead and a hypothetical single point of failure [11].

In the next chapters, different solutions are provided in order to overcome the depicted weaknesses by exploiting the Edge Computing approach and Cloud-Edge Continuum environment.

In particular, the provided solutions presented in the thesis aim to solve two main classes of problems, above introduced, which are referred to, in the thesis, with two terms:

- *Security*. It concerns the security issues in Cloud Computing
- *Distribution*. It concerns the centralization approach issues.

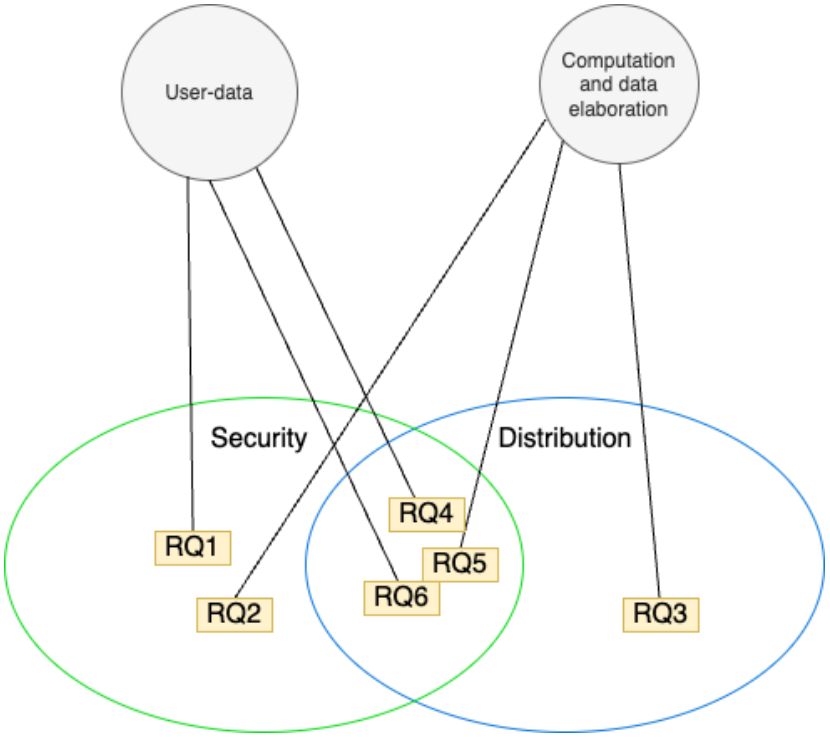

Figure 1.1: Research Questions Topic Organization

Figure 1.1 depicts how the research questions are organized exploiting the two terms introduced in our taxonomy and distinguishing three distinct categories: *Security*, *Distribution*, and the interplay between them. The categorized research questions, as shown, treat one of two aspects:

- The aspect related to data management, that the figure depicts with the circle *User-data*.
- The aspect related to computation management, that the figure depicts with the circle *Computation and data elaboration*.

Security issues in Cloud Computing

- **Question 1:** How Data can be stored and managed safely in a Cloud-Edge Continuum context?
- **Question 2:** How Data can be elaborated in Cloud Computing to protect the privacy of data owners?

Distribution of Cloud Computing centralized approach

- **Question 3:** How can complex computation like Machine Learning be figured out in the post-Cloud Computing era?

Distribution and Security intersected issues

- **Question 4:** How can User Identity Management be safely managed in Cloud Computing ?
- **Question 5:** How can Edge Computing and the decentralized approach improve User Identity Management?
- **Question 6:** How can the computation overcome the Cloud centralized approach to satisfy current applications?

1.1 Scientific contributions

This Section summarises the scientific contributions provided by this thesis, referencing the research questions already introduced.

Security issues in Cloud Computing

Question 1: How the Data can be stored and managed safely in a Cloud-Edge Continuum context?

Contribution 1: The security of data storage in Cloud Computing has consistently ranked as a paramount concern, as extensively documented in prior research [12]. While Cloud Computing offers compelling advantages in scalability, accessibility, and cost-efficiency, it presents inherent challenges that can potentially compromise the confidentiality and integrity

of sensitive information. A primary concern revolves around the risk of unauthorized access to stored data. Cloud service providers manage data on remote servers, leading to a relinquishment of control over physical access. This situation raises apprehensions regarding the sufficiency of security measures implemented by these providers and the potential for data breaches. In this context, our thesis explores this critical aspect. Specifically, a solution is designed to address the security weaknesses associated with Cloud Computing, as previously outlined. The proposed architecture leverages Edge Computing and Transparent Data Encryption to bolster the security of stored data, effectively mitigating the vulnerabilities inherent in conventional Cloud computing management practices.

Question 2: How the Data can be elaborated in Cloud Computing to protect the privacy of data owners?

Contribution 2: The management of data in Cloud Computing is intricately linked to the approach used for data processing. Encryption alone may not suffice when data undergoes various forms of processing. In the conventional approach, where computation is centralized within Cloud Infrastructure, data encrypted using traditional methods must be decrypted for processing, creating a temporal window during which they become accessible to the Cloud Infrastructure and the Cloud provider. This challenge has been addressed by embracing the Edge Computing approach and implementing Homomorphic Encryption. These measures aim to ensure secure data processing across different use cases, decoupling the data processing from reliance on specific Cloud Infrastructures and enhancing user data security. Through the combined application of these techniques, Edge Computing retains access to plaintext data while Homomorphic Encryption enables secure data processing within the Cloud, all while the data remains encrypted.

Distribution of Cloud Computing centralized approach

Question 3: How can complex computation like Machine Learning be figured out in the post-Cloud Computing era?

Contribution 3: In recent years, ICT research has increasingly focused on the decentralization of complex processes, moving away from the confines of traditional Cloud Computing Infrastructure. This shift is primarily motivated by the limitations inherent in the Cloud paradigm, which can compromise both security and computational efficiency. As a response

to these limitations, alternative approaches that, in various ways, complement or replace Edge Computing have been introduced. Particularly noteworthy is the extensive research conducted by numerous scholars who have explored solutions for the deployment and management of intricate processes within the realm of Edge Computing. This emerging approach has gained significant momentum, driven by the remarkable growth in computational resources offered by Edge Computing Devices in recent times. The advantages of this approach are manifold. It allows for the efficient utilization of bandwidth, ensures faster response times due to the proximity of raw data collection, enhances scalability possibilities, and bolsters the privacy and security of managed data. The proposed contribution aims to investigate an efficient alternative concerning the current literature. In particular, according to our approach, Machine Learning which, in current literature, has found a home within Edge Computing paradigms, can perform harmoniously between Cloud and Edge performing efficiently specific inference tasks.

Distribution and Security intersected issues

Question 4: How can User Identity Management be safely managed in Cloud Computing?

Contribution 4: Digital Identity plays a key role in modern ICT user services and even more secure approaches have been introduced in recent years. Indeed, each final Information and Communication Technology (ICT) service provided to a set of final users involves processes of Authentication, Authorization and Accounting (AAA) to ensure the authentication of the user, the management of roles and permission and the tracking of each activity performed by the final authenticated user. In order to enhance authentication, for example, Two-Factor Authentication (2FA) has been introduced in order to avoid the exchange of people adding a new form of identification concerning the classical single approach (password). This solution is usually solved using a one-time password (OTP). In Cloud Computing, privacy is not ensured; for this reason, it is correct to analyze the OTP management in a centralized Cloud Computing approach. In the following chapters of the thesis, a concrete solution for the safe management of 2FA via Cloud Computing and Microservices paradigm is investigated. The results show us a secure solution for managing OTP and mitigating the weaknesses of 2FA current implementations showing interesting time execution of the proposed protocol.

Question 5: How can Edge Computing and the decentralized approach improve User Identity Management?

Contribution 5: The question of Identity Management has long been explored in the context of a centralized approach, as evidenced by the literature. Historically, Cloud Computing and centralized architectures have been utilized to implement Identity and Access Management (IAM) [13]. While this approach offers advantages, it introduces limitations associated with a single point of failure and accessibility challenges when network resources are constrained. Indeed, centralized identity management can pose significant issues, particularly in rural areas where global network connectivity may be unreliable or limited. To address these challenges, an innovative decentralized approach has emerged. This novel paradigm leverages Edge Computing to facilitate the replication and distribution of Identity Management, thus enabling decentralized authentication and authorization. This approach ensures the protection of distributed resources, even in areas with network constraints.

Question 6: How can the computation overcome the Cloud centralized approach to satisfy the security and distribution constraints of current applications?

Contribution 6: The evolution of computation, including Machine Learning, is driven by two main factors: security and the elimination of the single point of failure in traditional Cloud Computing. This shift towards decentralization leverages Edge Computing, distributing computation across devices and servers. Edge Computing reduces data transit, enhances data privacy, and increases network resilience. It optimizes resource usage and enables real-time processing, mitigating the risks associated with centralized models. In essence, this revolution prioritizes security, efficiency, and reliability, shaping the future of computation in an interconnected world. For this reason, different techniques for secure data elaboration are investigated like Federated Learning. Several studies are deep in the thesis related to this new approach both in a centralized and decentralized fashion. The contribution introduced in our work tries to optimize Federated Learning techniques through a clusterization approach among Federated Learning clients improving the local computation of each involved device.

1.2 Structure of the Thesis

In Chapter 2 the basic concepts exploited in the rest of the thesis are described. In Chapter 3 and 4 the Security issues in Cloud Computing are seen in detail and different solutions for data management in security are proposed. In particular, these chapters propose a solution to manage securely data storage and elaboration in Cloud Computing thanks to the use of

the Edge Computing paradigm. In both cases, Edge Computing is considered the property of final users ensuring the data are never in plaintext within Cloud Computing Infrastructure. In Chapter 5 the Centralization of Computation is studied and a solution for the Artificial Intelligence inference in the Cloud-Edge continuum is proposed. In particular, exploit the Cloud-Edge Continuum Infrastructure and the Workflow concept, a distributed inference is designed and implemented. The proposed solution uses *Pegasus WMS*[14] to implement a Workflow that performs a distributed inference between Cloud and Edge infrastructure. Chapter 6, 7 and 8 will deep the third research questions group, about the Security and Distribution issues of Cloud Computing, investigating both the data side and computation side, as already specified in the previous section. Chapter 9 concludes the thesis with a recap of the problems and the solutions provided focusing also on future research works.

CHAPTER 2

Background

This chapter provides background information about technologies used as the basis of this thesis, in particular the Computing Paradigms involved in the thesis.

2.1 Cloud Computing

Cloud Computing is a paradigm of service delivery offered *on demand* by a provider to an end customer via the Internet, from a set of remotely available resources in the form of a distributed architecture. The strength of the Cloud is the fact that it resides on remote infrastructure (the Internet); users can access it from anywhere with a connection and a browser. Cloud services tend to mask the details of the underlying infrastructure from users, providing a simple graphical interface or API. Users do not need to be aware of internal system changes or configurations.

Cloud computing is based on two fundamental phenomena:

1. Pay-per-use model: Users pay according to the services they use and/or the time of use; the service adapts to the user's needs.

2. On-demand services: Services are always available and provided when needed.

When discussing Cloud Computing, it is essential to distinguish between the Public Cloud and the Private Cloud:

- Public Cloud: In this model, an external company, a.k.a. service provider, offers services to organizations. Examples of Public Clouds include, among others, Windows Azure, Google App Engine, and Amazon Web Services.
- Private Cloud: Here, the entity creating the Cloud service is the same one using it. A company can utilize its infrastructure to virtualize resources and provide them within the organization. However, this approach sacrifices some of the transparency, as it involves managing hardware and virtualization. On the plus side, it offers optimal and personalized resource management and improved service confinement, which enhances security. This solution is suitable for organizations prioritizing user privacy.
- Hybrid Cloud is a mixed solution in which the services are provided combining Public Cloud and Private.

The commonly implemented solutions are 'hybrid' models, where organizations combine private and public clouds into a single Hybrid Cloud. This approach offers consistent services, making the boundary between private and public services transparent to users. Another hybrid approach is the concept of a cloud federation or InterCloud, where multiple providers collaborate to offer a package of services. Users can choose a provider based on their needs, with agreements that may include mutual resource support among providers. Unlike the hybrid cloud, where a single provider combines its cloud with other public clouds, cloud federation relies on cloud solutions distributed across multiple providers.

The common denominator across these models is the delivery of features 'as a Service,' which aligns with the pay-per-use model and on-demand service.

Services in Cloud Computing are typically categorized into three *levels*:

- (SaaS) Software as a Service: Applications are offered as services, and end-users are the customers. Users no longer need to manage the software on their infrastructure, including updates and maintenance. They can simply use a browser as the computation is shifted to the server. An example of SaaS is Google Docs.
- (PaaS) Platform as a Service: This layer includes distributed systems installed within virtual machines. The systems are elastic, leveraging the underlying IAAS layer. PaaS provides high-level Application Program Interfaces (APIs) for developers to build cloud web applications on these infrastructures. Key characteristics of PaaS include deployment on virtual machines, provision of APIs for application development at a higher software layer, and an execution environment for applications.

- (IaaS) Infrastructure as a Service: This level refers to the use of virtualized resources, including hardware components such as storage services and processing nodes like virtual machines. Virtualization technology is employed, considering multiple interconnected physical servers with virtualization software installed. The hallmark of a cloud service is the complete abstraction of physical hardware, entirely managed by the cloud provider. An example of IaaS is a storage service like Amazon EC2 service which is the Amazon service that provides remote virtual machines.

In the context of Cloud Computing, it's essential to consider a configurable pool of computational resources. This refers to on-demand services accessible via the Internet and services provided by providers such as Google. As a result, applications can be provided by traditional computers or applications that utilize storage services and databases. Real servers can be interconnected with storage systems or computers linked together. The key distinction is that these components are virtualized, allowing services to be implemented by optimizing the real resources through the virtualization technique that is the heart of Cloud Computing.

2.2 Edge Computing

Edge Computing is a transformative paradigm in the realm of information technology and network architecture. It has emerged as a powerful response to the evolving demands of the digital age, characterized by the exponential growth of data, the proliferation of Internet of Things (IoT) devices, and the increasing need for low-latency, real-time processing.

Traditionally, the cloud has served as the hub for data storage and processing, with data travelling from devices or sensors to centralized datacenter. While this architecture has proven effective for numerous applications, it comes with inherent limitations, particularly in scenarios that require rapid decision-making, such as autonomous vehicles, industrial automation, and augmented reality applications.

The Edge Computing approach tries to overcome Cloud Computing limitations by offloading computation and data storage closer to the place where data are collected, typically at the network's edge or on IoT devices themselves. By doing so, Edge Computing reduces the time it takes for data to travel to a centralized Cloud infrastructure and return with a response. This reduction in latency is very important for applications where split-second decisions are critical, such as self-driving cars avoiding obstacles or industrial machines responding to safety concerns.

Moreover, Edge Computing enhances privacy and security. Since data processing can occur locally or near the data source, sensitive information can remain closer to the point of origin, reducing the risk of data exposure during transit to the cloud. This is particularly important in fields like healthcare and finance, where data security and privacy are paramount.

The Edge Computing landscape continues to evolve rapidly, driven by advancements in hardware, networking, and software technologies. It involves an ecosystem of edge devices, gateways, fog computing, and cloud integration, each playing a specific role in optimizing data processing and management. Edge Computing is increasingly influencing a wide range of industries, including healthcare, manufacturing, smart cities, and agriculture.

As Edge Computing continues to gain traction, researchers and industry experts are exploring innovative use cases, developing standards and protocols, and addressing the associated challenges, such as device heterogeneity, resource constraints, and security. The convergence of Edge Computing with emerging technologies like 5G, artificial intelligence, and blockchain promises to unlock new possibilities and revolutionize how data is processed, creating a more responsive, efficient, and secure digital environment.

2.2.1 Cloud-Edge Continuum

The cloud-edge continuum is a distributed computing paradigm that spans from centralized cloud datacenters to edge devices, such as IoT sensors and actuators. It enables data processing and application execution to be distributed across the continuum, based on factors such as latency, bandwidth, and security requirements.

The cloud-edge continuum is becoming increasingly important as the world becomes more interconnected and data-driven. It is essential for enabling emerging applications such as artificial intelligence, machine learning, and real-time analytics.

2.2.2 Benefits of the Cloud-Edge Continuum

The key benefits of the cloud-edge continuum include:

- Reduced latency: By processing data closer to the source, the cloud-edge continuum can reduce latency and improve performance for latency-sensitive applications.
- Improved scalability: The cloud-edge continuum can be scaled to meet the needs of even the most demanding applications.

- Enhanced security: The cloud-edge continuum can help to improve security by processing and storing sensitive data closer to the source. Reduced costs: The cloud-edge continuum can help to reduce costs by offloading processing tasks from edge devices to the cloud.

2.2.3 Challenges of the Cloud-Edge Continuum

The cloud-edge continuum is still in its early stages of development, but it is rapidly evolving. Several research projects and industry initiatives are underway to develop new technologies and standards to support the cloud-edge continuum.

One of the key challenges facing the cloud-edge continuum is the need for efficient and scalable data management. Data needs to be able to be moved seamlessly between the cloud and the edge, without sacrificing performance or security.

Another challenge is the need for effective application orchestration. Applications need to be able to be deployed and managed across the cloud-edge continuum in a coordinated manner.

The main challenges just described Cloud-Edge Continuum are deep in the thesis, as we'll see in detail, in order to understand how can make this new approach efficient in particular we'll see different solutions that will exploit it to improve the Cloud classical approach. Indeed, in Chapter 4, as we'll see in detail, we try to improve the security of the Cloud by exploiting a Cloud-Edge infrastructure and considering it a secure environment. In that case, a particular problem could be represented by data management between cloud and edge. In that case, data are encrypted and their management could represent one of the considered challenges in the Cloud-Edge continuum.

2.3 Homomorphic Encryption

The classical cryptographic systems have a notable weakness: encrypted data must be decrypted for practical use. While it's possible to store data in the cloud encrypted using secret keys, any operations requiring data manipulation, such as editing a text document or searching a database, necessitate decryption, potentially making the data vulnerable.

To address this challenge, modern cryptography has introduced a groundbreaking concept known as Homomorphic Encryption. Homomorphic Encryption is a cryptographic technique that allows the manipulation of encrypted data. It enables operations on encrypted data without the need for decryption. For instance, given two numbers A and B, both en-

crypted using the same homomorphic algorithm, it is possible to compute the encryption of their sum directly, without the requirement to decrypt the data. This approach ensures data remains protected and confidential even during computational processes in potentially insecure or untrusted environments, such as the public cloud. Homomorphic encryption is divided into two primary families:

- **Partially Homomorphic Encryption (PHE):** This type of encryption can process only one type of operation, typically addition or multiplication. Partially homomorphic encryption supports a select number of mathematical operations, allowing unlimited execution of either addition or multiplication on ciphertexts.

- **Fully Homomorphic Encryption (FHE):** FHE can process all essential operations, including arithmetic operations and Boolean functions like AND, OR, and NOT. Fully homomorphic encryption supports both multiplication and addition an unlimited number of times. This type is highly sought after as it ensures a high level of privacy. It permits the use of encrypted data for useful operations without the need for decryption. One of the most promising applications is enhancing the security of Cloud Computing.

The fundamental distinction between "Standard" Encryption and Homomorphic Encryption lies in the necessity to first encrypt and then decrypt data using a cryptographic key for analysis in the former. If the cryptographic key is compromised, the data becomes decipherable. In contrast, Homomorphic Encryption offers superior data protection and privacy. It enables data analysis without decryption.

Homomorphic cryptography derives its name from the mathematical concept of 'algebraic homomorphism.' In abstract algebra, a homomorphism is a mapping between two algebraic structures of the same type that preserves the operations defined within them.

This groundbreaking technique was conceived in 2009 by the American computer scientist Craig Gentry. It represents a form of encryption that permits data analysis without requiring access to the secret encryption key. In essence, homomorphism denotes a transformation applied to a data structure that generates a new one while preserving the relationship between elements in both sets. The ability to work with encrypted data rather than decrypted data offers a significant advantage, allowing data analysis and manipulation without exposing it to unauthorized parties.

Homomorphic Encryption is a promising avenue in the realm of data security, paving the way for a future where privacy and usability coexist seamlessly.

2.3.1 Homomorphism

In abstract algebra, the Homomorphism is defined as an "association" between two algebraic structures. An example of algebraic structure is the group.

A group consists of a set A along with an operation associated that we can indicate with the symbol '+' (called the group law of A) that joins any two components x and y to frame another component, named $x + y$.
According to linear algebra, to define the ensemble of set and operation named *(A,+)* like group, they have to satisfy the following requirements:

- Closure: For any x and y taken from A, the result of the operation $x + y$ belongs to group A

- Associativity: For any x, y, z belonging to group A we have:

$$(x + y) + z = x + (y + z) \tag{2.3.1}$$

- Identity element: In the group A, we mandatory have an identity element e such that for each element x in the group A, the following relation is true:

$$x + e = e + x = x \tag{2.3.2}$$

- Inverse element: In the group A, for each x element, there exists a y element such that:

$$x + y = y + x = e \tag{2.3.3}$$

  where e is the identity element aforementioned.

The identity element inside a group is often indicated with 1. An important consideration to mention is related to the commutative property of the operation '+'. This is not obvious in a group. Indeed a group, in which the group law's commutative property is valid for any element of the set, is named *abelian group*.
Given two groups *(A,+)* and *(B,*)*, we can define a group homomorphism from *(A,+)* to *(B,*)* like a function

$$f : A -> B \tag{2.3.4}$$

such that for each a and a' that belong to G we have:

$$f(a + a') = f(a) * f(a') \tag{2.3.5}$$

In this definition, we can already see the basic principle of Homomorphic Encryption. Indeed, the definition describes an equation in which the image of a function of an elaborated value is equal to the elaboration of individual images. This equality could be easily translated into the Cryptography environment.
If we consider an Encryption Scheme where P and C are respectively the Plaintext and the Ciphertext spaces and if we imagine that P and C are groups, defined as $(P, +)$ and $(C, *)$ we can say that Homomorphic Encryption algorithm is a function E from P to C by which for all a and b in P the following equality is true:

$$E(a) + E(b) = E(a * b). \tag{2.3.6}$$

Paillier Algorithm

One of the Algorithm exploited in this work is the Pailler Algorithm. The Pailler Algorithm is a very famous Homomorphic Encryption Algorithm formulated, for the first time, by Pascal Paillier. It is a probabilistic public-key algorithm and it's based on the problem of computing n-th residue classes, and in particular, when the module is $n = 2$, that is considered computationally difficult. The algorithm was firstly described on [15].
If we consider a prime number p, a generic number $a \in \mathbb{Z}$ is defined a module p n-th residue if the following algebraic congruence has a solution:

$$x^n \equiv a \mod p \tag{2.3.7}$$

One of the novelties of the algorithm designed by Pailler is that in this specific problem the module used is a square number and in particular it's n^2. Hence, the congruence is defined:

$$z \equiv y^n \mod n^2 \tag{2.3.8}$$

In this case, if y exists and if it belongs to $\mathbb{Z}^*_{n^2}$, z is described as a n-th residue modulo n^2. The author proves that checking if an element is an n-th residue is, as we have already said, a

difficult problem. Starting from this congruence, Paillier has designed a function named ϵ_g:

$$\mathbb{Z}_n \times \mathbb{Z}_n \longrightarrow \mathbb{Z}^*_{n^2} \tag{2.3.9}$$

$$(x, y) \longrightarrow g^x * y^n \mod n^2 \tag{2.3.10}$$

with g that is a random element of $\mathbb{Z}^*_{n^2}$.
As we'll see deeply, the function ϵ_g will represent the encryption function within Paillier scheme.
Moreover, Paillier defines in his work the function L in the subgroup

$$S_n = \{u < n^2 | u = 1 \mod n\} \tag{2.3.11}$$

defined as

$$\forall u \in S_n \tag{2.3.12}$$

$$L(u) = \frac{u-1}{n} \tag{2.3.13}$$

At this point, we are going to see, in detail, the steps involved in the key-generation:

- Select two large prime numbers p and q such that $gcd(pq, (p-1), (q-1))$.
- Compute $n = p * q$ and compute the Carmichael function that in the specific case is: $\lambda = lcm(p-1, q-1)$
- Select a random number $g \in \mathbb{Z}^*_{n^2}$. It's the number already introduced in the function ϵ_g
- The public key and the private key are respectively (n, g) e (p, q)

At this point, exploring the key we can see the encryption and decryption algorithm. Let m be a message to be encrypted where $0 \leq m < n$. Select random r where $0 < r < n$ and $\gcd(r, n) = 1$.

Compute ciphertext as:

$$c = g^m \cdot r^n \pmod{n^2}$$

Let c be the ciphertext to decrypt, where $c \in \mathbb{Z}^*_{n^2}$.

Compute the plaintext message as:

$$m = \Big(L(c^\lambda \pmod{n^2}) \cdot \mu\Big) \pmod{n}$$

2.4 Federated Learning

Federated Learning is a novel paradigm that shifts the traditional model of centralized data processing to a more collaborative and privacy-aware framework. Unlike conventional machine learning, where data is centralized and aggregated for model training, Federated Learning enables model training across decentralized and distributed data sources.

This innovative technique allows devices, edge nodes, or even different organizations to collaboratively train machine learning models without the need to share sensitive data. By maintaining data on local devices and only sharing model updates, Federated Learning provides a crucial solution for preserving user privacy and data sovereignty.

The applications of Federated Learning are diverse and impactful, ranging from personalized recommendation systems and predictive text input on mobile devices to healthcare analytics and industrial IoT. In this introduction, we will delve into the key principles, benefits, and real-world use cases of Federated Learning, highlighting how it is shaping the future of AI by combining the power of collaboration and data protection.

Federated Learning was introduced by Google with the paper [16] in which the first aggregation algorithm was introduced: FederatedAveraging.

The paper starts from the algorithm of Federated Stochastic Gradient Descent (FedSGD) that is used as a benchmark in FedAvg comparison. FedAvg is born starting from FedSGD but considering more than one epoch on the local client e considering an average of the trained weights in a Parameter Server.

In literature, two kinds of Federated Learning are mainly distinguished according to the topology used in the aggregation phase:

- Centralized Federated Learning that represents the *traditional* approach in which a central server, also named aggregation server, aggregates all the weights trained in local clients

- Decentralized Federated Learning represents an innovative approach in which a central server is not present and the aggregation of weights is performed through direct communication among the involved clients. This kind of approach is studied and several solutions are involved in this category. Indeed, very different solutions could be considered if a central server is not present [17]. The current state of the art, for example, considers two possible decentralized topologies:

  – A total decentralized approach in which the aggregation is performed in a dis-

tributed way.

- A semi-decentralized approach in which the aggregator server is eligible among the distributed clients.

Part I

Security Issues in Cloud Computing

CHAPTER 3

Data Storage Protection in Cloud-Edge Environment

Edge systems are increasingly popular for data collection and processing. Typically, due to their limited storage capacity, pieces of data are continuously exchanged with Cloud systems which store them in distributed DataBase Management System (DBMS). This scenario, known as Cloud/Edge Continuum, is critical from a data security point of view as it is exposed to many risks. Transparent Data Encryption (TDE) is proposed as a possible solution for encrypting database files. However, current solutions do not suit the Cloud/Edge continuum requirements. In this chapter, we aim at fulfilling this gap by proposing a solution to encrypt the data locally at the Edge and transfer them to a distributed database over the Cloud. Our approach allows us to perform queries directly on encrypted data over the Cloud and to retrieve them on the Edge for decryption. Experiments performed on different NoSQL DBMS solutions demonstrate the feasibility of our approach.

3.1 Problem Statement

Cloud Computing has been an innovative and widely used paradigm due to its considerable advantages. As described in previous chapters, Cloud Computing allows the end-user to efficiently manage Information and Communication Technology (ICT) resources in an easier way abstracting and providing them as Infrastructure, Platform, and Software services. The use of this approach was disruptive, but the abstraction provided to the end user has introduced all the discussed weaknesses in Chapter 1. The Edge computing paradigm has

emerged to partially mitigate the aforementioned Cloud issues. This innovative approach involves the use of tiny devices (e.g., microcontrollers and/or microprocessors) to process data closer to the place where it is generated and collected with the purpose to change the typical centralised Cloud approach. However, the Edge is not conceived to perform Big Data processing that, sometimes, has to be moved to the Cloud. Therefore, according to the specific system requirements, micro-services could be moved from the Edge to the Cloud and vice versa. This paradigm is referred as Cloud/Edge Continuum [18]. In this research work, we consider a Cloud/Edge continuum system scenario able to manage sensor logging Big Data in a secure fashion as shown in Figure 3.1. For this reason, a peer-to-peer, NoSQL document-oriented, and Transparent Data Encryption (TDE) enabled DataBase Management System (DBMS) is strongly required. Specifically, a DBMS for a Cloud/Edge Continuum environment must be: *peer-to-peer* because it has to be executed in both the Cloud and unreliable Edge devices; *NoSQL document-oriented* to simplify the management of sensor logging Big Data according to a schema-free approach; and *TDE-enabled* to store the file system in a secure fashion. Nevertheless, at the time of writing this thesis, a DBMS providing all these features does not exist. For example, by analysing the major NoSQL DBMS solutions, Cassandra is peer-to-peer and provides TDE, but it is column-oriented instead of document-oriented; MongoDB is document-oriented and provides TDE, but it adopts a master-slave instead of a peer-to-peer architecture; etc. To overcome all the aforementioned DBMS requirements,

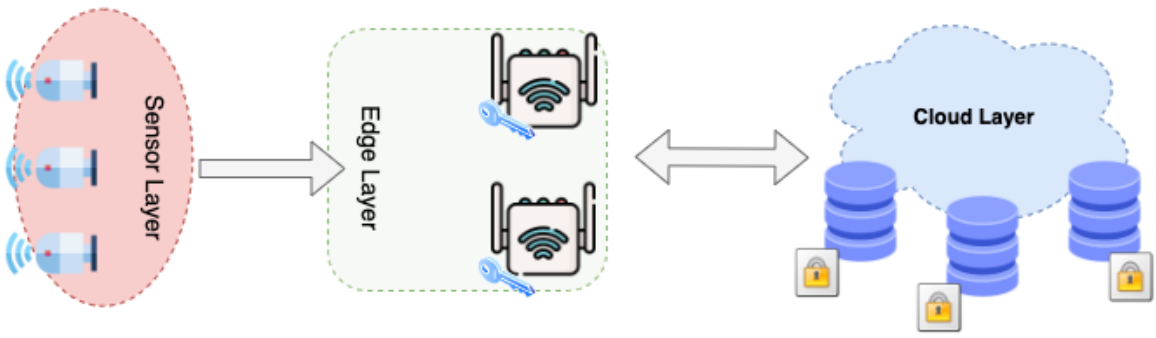

Figure 3.1: TDE Application: Reference Scenario.

in this chapter, we start with CouchDB, i.e., a NoSQL document-oriented DBMS adopting a peer-to-peer data distribution approach in which all the cluster nodes are synchronized even though they are not always up and ready. Specifically, it is easy to be deployed in a cluster in which it is not required the continuous connection of nodes that own a part of the whole data and that are synchronized when they are up and reachable. This approach is in contrast to other alternative NoSQL document-oriented databases such as MongoDB in which

the distribution of data is implemented through the master-slave approach. Nevertheless, CouchDB does not support TDE. Therefore, this chapter aims at overcoming such a gap. The remainder of the chapter is organized as follows. Section 3.2 presents the literature about TDE and in general about the security of data management at the Edge. Section 3.3 describes the high-level architecture of the proposed solution. A system prototype is described in Section 3.4. Experimental results are shown and described in Section 3.5. In the end, conclusions and future works are summarised in Section 3.6.

3.2 Related Work

Nowadays, the number of Cloud/Edge continuum scenarios (e.g., in smart cities, healthcare, finance, etc.) is continuously increasing. Data security is one of the most crucial aspects to focus on when developing Cloud/Edge systems. In particular, considering Edge computing, data vulnerability is ever greater [19]. Various aspects concerning the security of Edge and/or Internet of Things (IoT) devices have been addressed in the literature [20], referring to complex certification processes [21] from various points of view. Sometimes, especially for large-scale deployments, expensive solutions in terms of technology and/or business are not feasible for companies [22]. It is necessary to implement actions that allow secure data collection and exchange in Cloud/Edge systems even in the case of large deployments how, for example, in a smart city where thousands of Edge devices are scattered around it and they exchange data with each other ones and with Cloud systems [23]. In this case, it is possible to think about encrypting the data stored in such systems. In [24], the authors focus on the positive aspects of using TDE on a standard database by assessing how good security performance degrades system performance. The TDE is often used to improve the security of relational DBMS(s) [25]. In [26] the authors remark that using this technique the database will hit a CPU and storage performance overhead. However, on Cloud systems, this situation can be managed. The authors demonstrate that the impact of managing security aspects can be optimized by considering CPU, I/O, and RAM performance. The purpose of our paper is to use this technology in a Cloud/Edge continuum scenario by evaluating its performance. With the spread of Big Data, security has become the major issue. In addition to concerning the acquisition systems [27], it is a problem of archiving software. In [28] an analysis is done to evaluate how the open-source Hadoop Distributed File System (HDFS) is used to store huge amounts of data with high throughput and fault tolerance. However, the security model was not designed and has become the main drawback of Hadoop. In terms of

storage, metadata and data security is a problem for HDFS. The authors in their work explain the importance of this issue by explaining how companies are showing only a few layers of security in Hadoop such as Kerberos and TDE which is, therefore, a technology also used in the corporate world. These problems are now common also in Edge computing and, for this reason, we want to investigate the use of TDE in Cloud/Edge continuum environments.
In [29] the authors focus on how relational databases provide built-in security controls and mechanisms. The problem, which is most evident in Edge environments, is that information residing in the data store is at great risk. In their work, the authors try to introduce a new level of security. The used approach consists of segregating information based on its level of sensitivity and dynamically creating referential integrity constraints during the execution. For example, the primary keys of the restructured tables and the attributes of the most critical information were protected using the TDE utility provided by Oracle to prohibit the illegitimate use of information. The authors measured the querying performance of this approach. Our work aims at improving performance even while maintaining a solution of the same type. In [30] the authors note that the solutions for TDE provided by the major DBMS solutions are limited to protecting data at rest only and appear to be useless if the adversary has physical access to the server, which is a likely risk during the hosting at the Cloud or at the Edge. The authors propose an alternative approach to TDE based on an abstract model. This model takes into account the specific risks of the Cloud and extends the encryption to cover data in use and partial data in motion. Our research work, instead, aims at managing data in motion in a secure way.
The data security issue is addressed in [31], where the authors analyse and compare two current approaches: Oracle's TDE and the standard encryption provided by MySQL. Also interesting is the assessment that is made considering both single-server and distributed configurations. We intend to go beyond the approach of using TDE for MySQL databases and try to understand which technology is more functional in Edge/Cloud scenarios.
In fact, the work proposed in this paper aims at introducing an innovative solution that uses the AES algorithms [32] to encrypt data without degrading system performance even in Edge environments [33, 34].

3.3 System Architecture

Figure 3.2 shows the system architecture designed for secure storage of sensor logging Big Data through TDE mechanisms in a Cloud/Edge continuum context. The architecture

is divided into three macro-layers to allow a more detailed analysis of each element and its features.

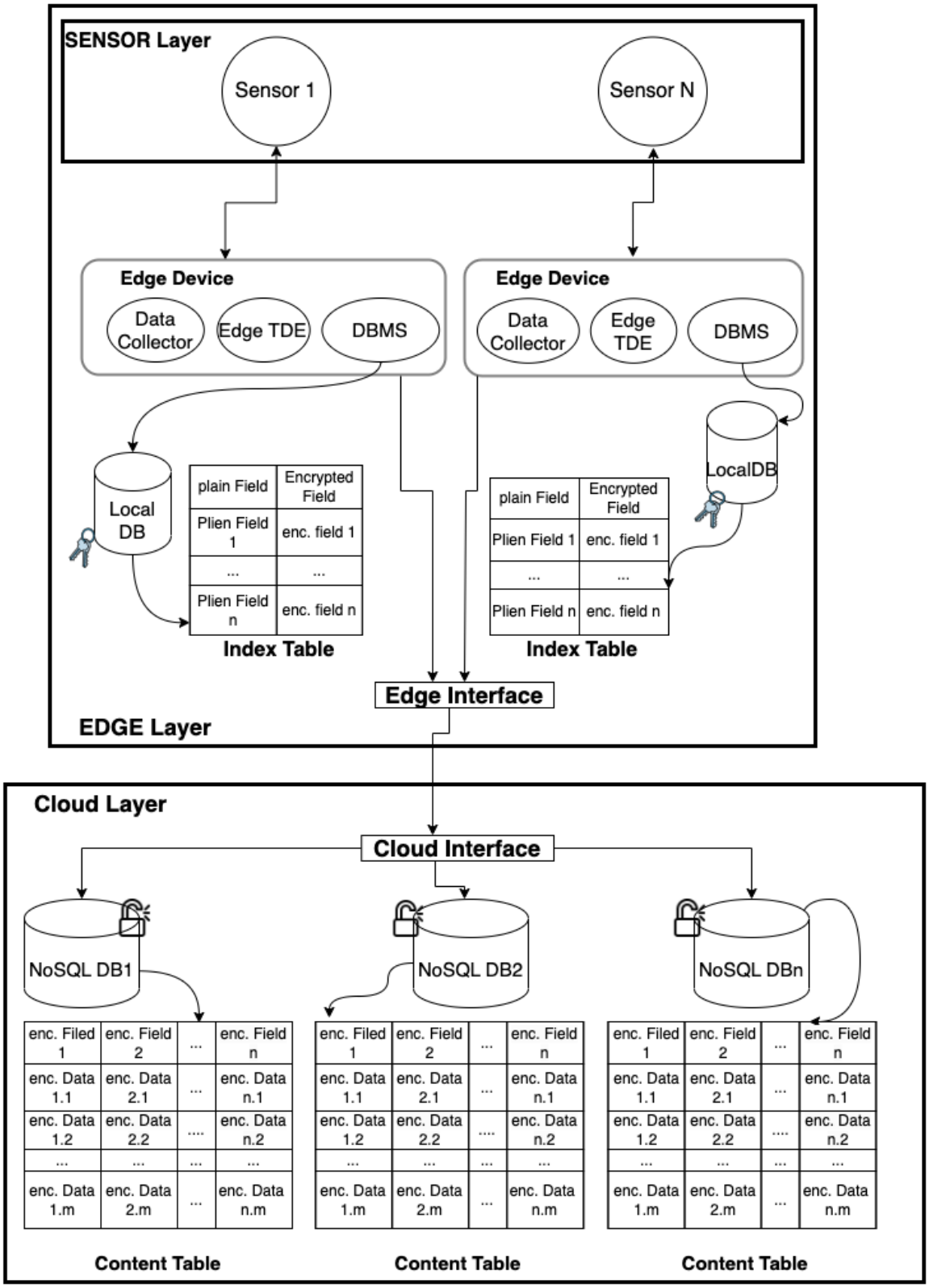

Figure 3.2: System Architecture.

The *Sensor layer* is characterized by sensors that detect and collect data from the environment in which they are installed. Every single sensor generates hundreds of logging data that are managed by the Edge layer.
The *Edge layer* includes microprocessor and/or microcontroller devices able to acquire and process the data generated by the sensors acting in the Sensor Layer. Inside the Edge layer, there are different software modules:

- *Data Collector* is responsible to collect and process pieces of data coming from sensors.
- *Edge TDE* is responsible to start the TDE mechanism to encrypt the data acquired by the Data Collector;
- *DBMS* acts as an interface between the acquisition and storage system. The data acquired by the sensors are encrypted within the Edge Layer and transmitted to the Cloud system. Every single Edge device is equipped with a symmetric key used to encrypt the acquired data, and a Local database.

The *Cloud Layer* refers to all the software components running in the Cloud, which enable communication with the Edge devices of the Edge Layer. The Cloud only serves as a storage for encrypted data. In this way, any malicious user who attacks Cloud Storage systems can only recover the encrypted values and not obtain information on the adopted data model.

Particular attention has been paid to the design of TDE mechanisms for the secure archiving of logging sensor data in a Cloud/Edge Continuum context based on a NoSQL Database. The main idea is to memorize all the logging sensor data collected at the Edge into a Cloud storage system by adopting a TDE approach. As previously stated a NoSQL document-oriented, schema-free and peer-to-peer DBMS solution supporting TDE currently does not exist. For this reason, we focused on the design of TDE mechanism for a distributed database in a Cloud/Edge continuum environment. In the following, we describe how our TDE works. Every single sensor sends the plain data to the Edge device where the data encryption mechanism is started. The TDE uses the symmetric key encryption approach based on the AES protocol to encrypt data related to both field names and contents. Nevertheless, we had to face an issue paradoxically caused by the robustness of the AES protocol: if the same piece of data is encrypted twice with the same key, the result is different due to the presence of a random factor that allows increasing the security of data encryption. This means that if we encrypt a new field name in the Edge device and we store it in the distributed database

deployed over the Cloud and if we subsequently try to compose and execute at runtime a query on the Edge device, we obtain an error because the two encrypted fields names corresponding to the same plain field (the one encrypted at the time of data storage over the Cloud and the one encrypted subsequently at the time of query composition and execution at the Edge) are different. This prevents the execution of queries. To solve this problem it was decided to create a simple mapping system for encrypted field names using an index table stored in the local database of the Edge device including the list of plain fields and their corresponding encrypted data.
In this way, the data model for the acquired logging sensor data can be dynamically built according to a NoSQL document-oriented and schema-free approach within the Edge Layer and the actual encrypted data can be transmitted to the distributed database deployed over the Cloud. In the Cloud database, the data (both field names and contents) is saved encrypted and no decryption mechanism can be foreseen. In this way, a malicious user can only acquire meaningless encrypted data in the absence of a decryption key. In case the end-user wants to obtain the data stored in the Cloud distributed database, it is necessary to compose queries using the encrypted field names stored in the Edge device's local database. This mechanism enables the development of a robust Cloud/Edge continuum ecosystem in terms of security that can prevent the classic attacks made on the storage systems.

3.4 Implementation

In this Section, we describe in detail the implementation of the TDE mechanism for data encryption on the Edge and its storage on a NoSQL DBMS deployed in a Cloud system. We have focused our work on two fundamental phases: 1) *Data acquisition, encryption and memorization* and 2) *Data retrieval and decryption*.

3.4.1 Data Acquisition, Encryption and Memorization

As a NoSQL database, the choice fell to CouchDB, a NoSQL document-oriented, schema-free and peer-to-peer solution. The main problem with using CouchDB is the lack of an integrated TDE mechanism: our objective is to fulfill such a gap. The logging pieces of data that are acquired by sensors are sent from the sensor layer to the Data Collector software module acting at the Edge Layer (Figure 3.3 - Steps: 1 - 2)). The Data Collector allows the Edge layer to acquire plain logging sensor data and send it the Edge TDE module (Figure 3.3 - Step 3), which implements the TDE mechanisms. Within the Edge TDE module, the plain

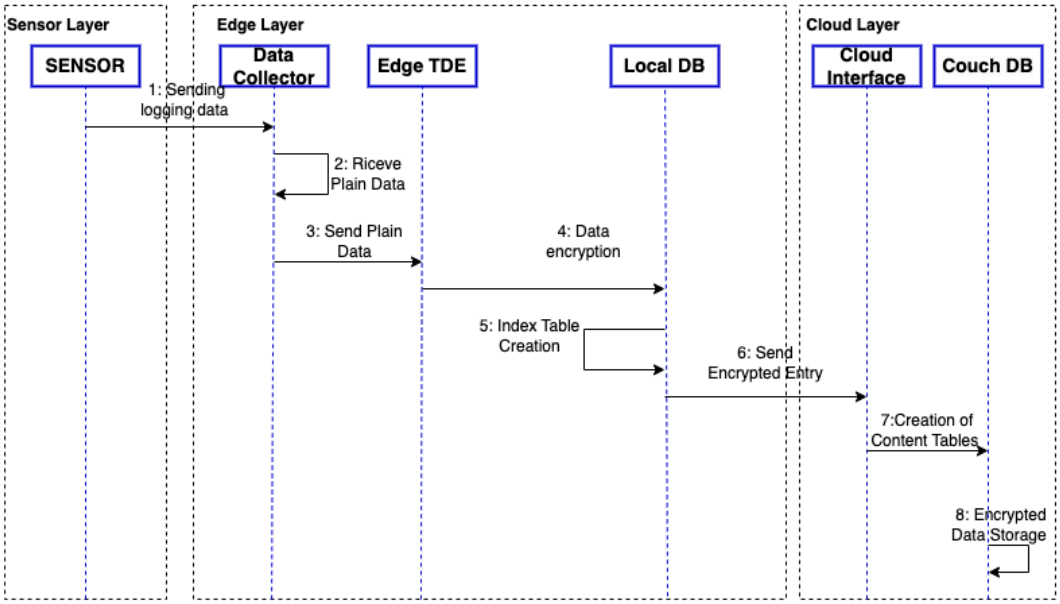

Figure 3.3: Data storage flow.

logging sensor data is encrypted using the AES 256 symmetric encryption algorithm and sent to the Local DB (Figure 3.3 - Step 4)). To solve the problem due to the random factor of this algorithm, it was necessary to create an index table (Figure 3.3 - Step 5) that is used to map the correspondence between the plain fields names and the corresponding encrypted data. The index table is created in the local database present within each single Edge Device. The encrypted data is transmitted to the Cloud Interface module acting at the Cloud Layer (Figure 3.3 - Step 6). This software module has been implemented to allow communications between the Cloud Layer and the Edge Layer, specifically allow the sending of encrypted data to Couch DB. To store the encrypted data on Couch DB it is necessary to create a Content Table containing both the encrypted fields names and also content data, i.e. the encrypted sensor logging data (Figure 3.3 - Step 7). By creating this table it is possible to save the encrypted sensor data on the Cloud (Figure 3.3 - Step 8).

This design choice was made to allow that sensor logging data can be saved in CouchDB in encrypted mode over the Cloud. As a consequence, in the event of an attack, the malicious user can access encrypted data. The mapping between plain fields name and corresponding encrypted data is contained within the Edge, therefore both encryption and decryption operations can be performed only at the Edge layer.

3.4.2 Data Retrieval and Decryption

For data retrieval and decryption, a query-based mechanism has been implemented at the Edge layer allowing secure data extraction from the Cloud. This design choice was made to allow an high level of security and reliability of the data. The authorized user can access

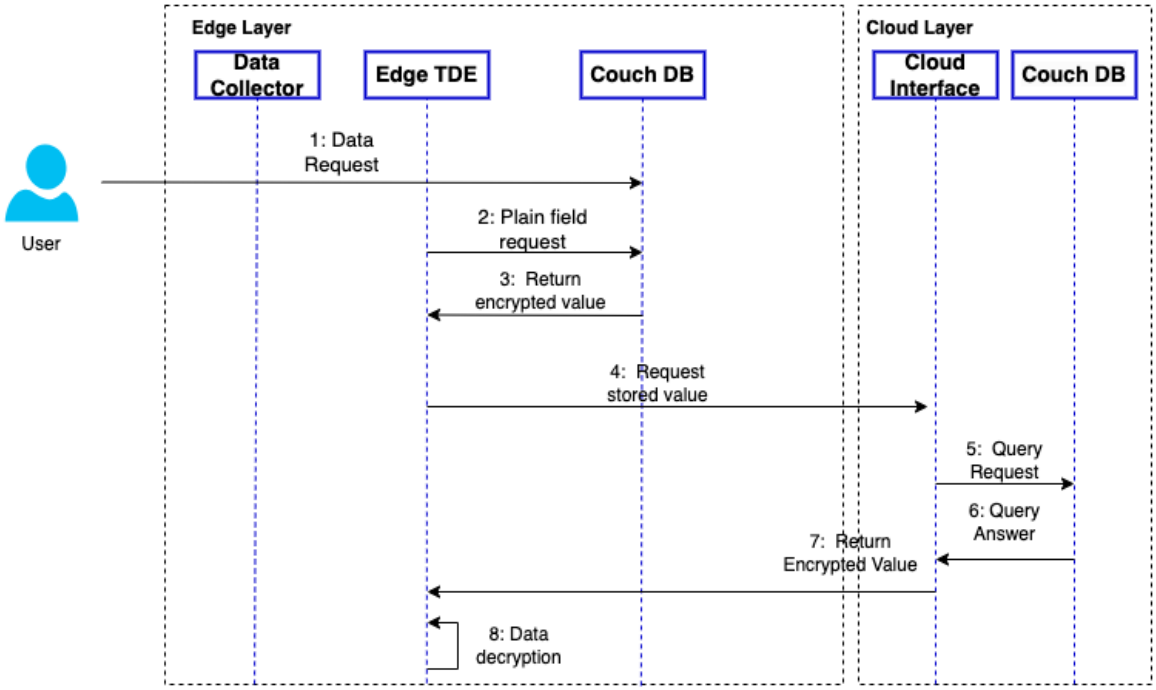

Figure 3.4: Data Decryption flow.

the Edge device and retrieve the data stored in CouchDB over the Cloud. The user interacts with the Edge Layer to start the data retrieval and decryption process (Figure 3.4 - Step 1). The Edge layer of Couch DB requests the local Database to extract encrypted fields name contained in the index table, (Figure 3.4 - Step 2). The encrypted fields names are returned to the Edge TDE software module where the data decryption processes take place. (Figure 3.4 - Step 3).

From the Edge TDE, a query is build considering encrypted fields names that is submitted to CouchDB acting at the Cloud Layer to extract the encrypted content data (Figure 3.4 - Step 4). In the Cloud layer, the Cloud Interface software module manages all the queries that are made from the Edge device to CouchDB. The Cloud Interface carries out the query on the CouchDB database which replies with the requested encrypted content data (Figure 3.4 - Steps: 5 - 6). At this point, having found the data requested by the user, the Cloud Interface sends the encrypted content data to the Edge TDE module running in the Edge device (Figure 3.4 - Step 7). The Edge TDE module contains the decryption symmetric key to decrypt the

content data that are returned to the user (Figure 3.4 - Step 8).

3.5 Experimental results

The experiments presented in this Section demonstrates the goodness of the described solution. In particular, a dataset containing logging sensor data, i.e., temperature, humidity, and ping times, was used. The dataset contains 130.338 records. Specifically, the dataset was divided into four parts of increasing size. A summary of the size breakdown of the considered dataset is reported in Table 3.1. The data has been encrypted and a dataset with

Dataset Dimension	Number of Records
25%	32.584
50%	65.169
75%	97.753
100%	130.338

Table 3.1: Experiments Dataset Partition

encrypted field names has been created. For performance evaluation, projection queries were made. Three queries of increasing complexity were considered allowing us to compare the performance of the system using three of the major NoSQL DBMS, that are Cassandra, MongoDB, and CouchDB:

- first query: returns all documents that have the ping value equal to 17.28 seconds;
- second query: returns all documents that have the ping value equal to 17.28 seconds and a recorded temperature value greater than or equal to 22.00 degrees Celsius;
- third query: returns all documents which have ping value equal to 17.28 seconds, recorded temperature value greater than - equal to 22.00 degrees Celsius, and humidity value greater than 35%;

For the sake of clarity, we underline that the traditional relational DBMS (RDBMS) terminology changes with the considered NoSQL DBMS solutions: the concept of record or tuple typical of an RDBMS turns into a row (considering Cassandra) or document (considering MongoDB and CouchDB). The concept of a table changes into a collection considering MongoDB and CouchDB and so on. Furthermore, the performance of a mechanism that allows regenerating the database containing the encrypted field names has been evaluated. The tests were carried out on a Virtual Machine (VM) with the following hardware characteristics:

- Microprocessor: AMD® Ryzen 7 3700U, 4 cores;
- RAM: 8 GB;
- Hard drive type and capacity: NVMe SSD 128GB

Linux Ubuntu 20.04.03 LTS was installed on the VM along with:

- MongoDB: enterprise edition, version 5.0.6;
- DataStax Enterprise (Cassandra): Version 6.8.20;
- CouchDB: Version 3.2.1.

The software modules have been developed using Python 3.8.10.

3.5.1 Querying Time Performance

The querying response times allow us to understand if the proposed system can be used in real Cloud/Edge Continuum scenarios. Three tests were performed with increasing query complexity. In Figure 3.5 the results of the first query are reported, it is evident that MongoDB

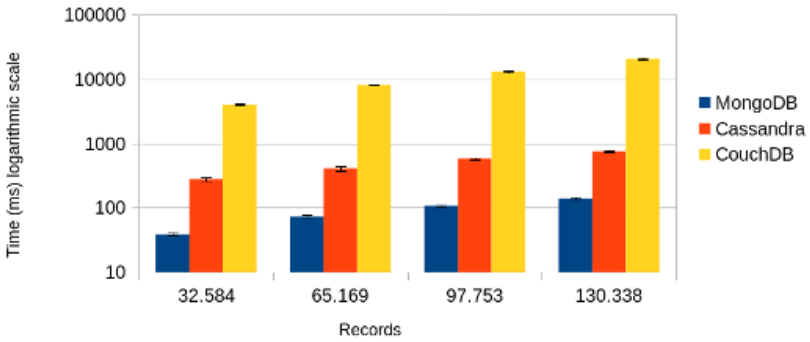

Figure 3.5: First Query - Medium Processing Time.

is the most efficient in terms of querying response times. In Figure 3.6 the results of the second query are reported. Also, in this case, MongoDB is the most efficient solution in terms of querying times. In Figure 3.8 the results of the third query are reported. Even in this case, MongoDB is the most efficient solution in terms of querying times. As highlighted, the results show MongoDB as the best-performing solution. It is equally evident that for the Edge context, the other solutions are also usable. However, the only solution that supports the described requirements (document-oriented schema-free data model, peer-to-peer architecture and TDE support) is CouchDB. The experiments, therefore, allow us to evaluate the price to pay for having a database solution fitting all the requirements of a Cloud/Edge continuum ecosystem which guarantees a high level of security through the TDE.

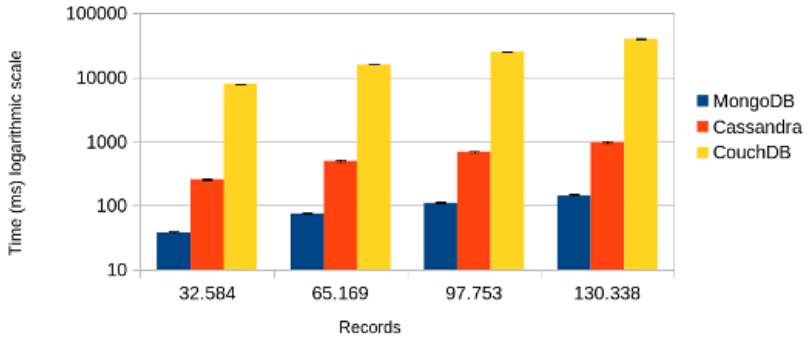

Figure 3.6: Second Query - Medium Processing Time.

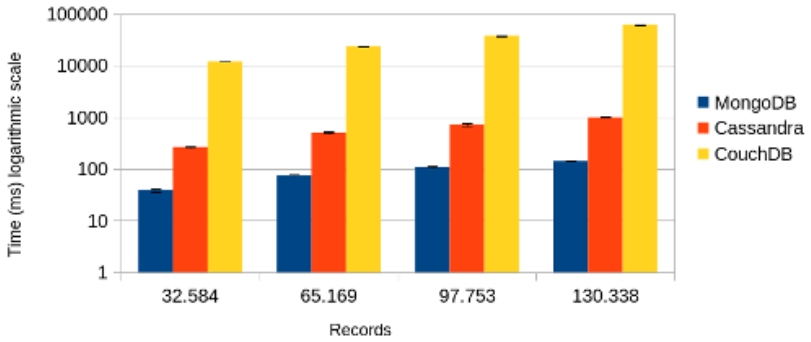

Figure 3.7: Third Query - Medium Processing Time.

3.5.2 Table Regeneration Time

The case study considered poses the problem of data availability. Edge devices are not only prone to attack scenarios but also prone to failure. If the index table is lost on the Edge device, it is necessary to regenerate it and update the content data collection in CouchDB over the Cloud. This is necessary due to the previously described behaviour of the AES algorithm. Such a procedure makes it possible to keep the system as a whole always operational. The

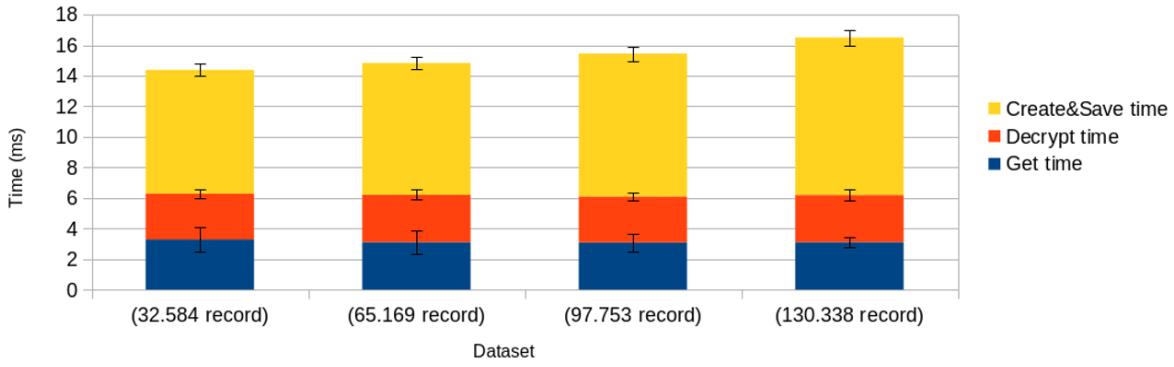

Figure 3.8: Time Keys Regeneration - Medium Processing Time.

experiments show that the time required for encrypted data regeneration, and therefore for the recovery of the system after an attack or a hardware failure in the Edge device, grows

linearly as the dataset grows. This result shows the efficiency of the implemented solution in terms of performance and therefore usability.

3.6 Remarks

The work described in this chapter allowed us to study, design and develop a system that would allow us to guarantee data security in a Cloud/Edge continuum system. The developed database system meets the requirements of the document-oriented schema-free data model, peer-to-peer architecture, and TDE support. This objective has been achieved with the use of CouchDB into which the TDE has been integrated. From performed experiments emerged that the TDE solution based on CouchDB is slower in terms of computational times than the other tested solutions. However, this result is compensated by the higher degree of fault tolerance that our solution manages to give to distributed Edge/Cloud continuum systems. We think that our solution can fit different application scenarios such as healthcare and financial ones in which security and privacy are crucial aspects. In future works, we plan to optimise our solution in terms of flexibility and processing time.

CHAPTER 4

Secure Elaboration in Cloud Computing via Homomorphic Encryption

For years, one of the major issues against the adoption of Cloud computing has been security. Typically, with traditional symmetric and asymmetric encryption schemes, data can be encrypted and sent into Cloud data centers, but this approach presents a great weakness: data must be transferred to the client-side to be deciphered and then processed. Nevertheless, such an approach is not feasible for a large amount of data. In this chapter, we address such an issue through the adoption of homomorphic encryption in a Cloud/Edge continuum environment. Specifically, pieces of data are encrypted in the Edge and transferred into the Cloud for storage and processing without the need to decode them. Experiments performed on different homomorphic encryption schemes in a Cloud/Edge testbed highlight the goodness of our approach.

4.1 Problem Statement

As mentioned in the previous sections, the Cloud Computing paradigm brings with it some known issues that can definitely compromise the security of managed data and consequently the privacy of end-users. The innovation and the business advantages are indisputable and we can say that Cloud Computing has totally changed the mindset by which Information and Communication services are provided. The transparency through which the resource is provided thanks to Cloud Computing allows end-user to easily access many services simply by using an Internet connection and a web browser exploiting the

pay-per-use paradigm. This approach is fundamental in the Cloud service delivery model named Software as a Service (SaaS). According to such an approach, the software is provided like a typical home utility and the end-user no need to worry about the update of the SaaS, the environment in which it is deployed, and how it is managed. The final user can simply utilize the SaaS just with an Internet connection and, usually, with a web browser. Moreover, with the advent of Internet of Things (IoT) an increasing amount of sensor data has been transferred and processed into the Cloud [35]. All these advantages have concrete consequences. The management of data is up to the Cloud Provider. However, the Cloud provider itself can represent a threat to the user's data. Typically, with traditional symmetric and asymmetric encryption scheme, data can be encrypted and sent into Cloud data centers, but this approach presents a great weakness: data must be transferred to the client side to be deciphered and then processed. Nevertheless, such an approach is not feasible for a large amount of data.
In this chapter, we address such an issue through the adoption Homomorphic Encryption (HE) in a Cloud/Edge continuum environment. Recently, Edge computing has emerged as a new distributed computing paradigm that allows moving data storage and processing closer to the source of that data to reduce the latency in data transmission of Cloud providers.

However, this is not always possible because some services needs to be decoupled in a Cloud/Edge Continuum environment. This is the case of secure massive data storage and processing. According to such an approach, pieces of data can be encrypted in the Edge and transferred into the Cloud for storage and processing without the need to a-priori decode them. Hence, According to our scenario, each tenant (e.g., government, hospital, bank, etc.) accesses the service through its trusted Edge layer in order to perform operation on its sensitive data. This is the only place in which plain text data are temporary stored. Once encrypted with an homomorphic scheme, data is sent to the Cloud provider for massive storage and processing. In the end, the result of processed encrypted data is sent back to the Edge layer. This advantage of our approach is twofold: a) no data is disclosed over the Cloud; b) it overcomes the psychical limits of the Edge layer about data storage and processing especially for the onerous computation task due to HE.
Furthermore, in order to study the impact of homomorphic encryption in encrypting data in the Edge and perform aggregation operation queries on encrypted data in the Cloud, in our experiments, we tested different libraries and different algorithms in order to evaluate the overhead introduced by Homomorphic Encription within the whole process. In particular we'll consider two libraries, both implemented in Python programming language. The first one implements the Pailler algorithm [15]. The second one exploits BFV algorithm

[36] and CCKS [37]. The remainder of this chapter is organized as follows. An overview about the current solutions regarding the secure elaboration of encrypted data according to Homomorphic approach is provided in Section 4.2. The designed system architecture is presented in Section 4.3. A prototype implementation considering different alternative HE libraries is discussed in Section 4.4. Experiments performed in a Cloud/Edge testbed highlighting the goodness of our approach and current limits are discussed in Section. 4.5. Section 4.6 concludes the chapter also providing light on possible solution improvement.

4.2 Related Work

The first Encryption Scheme is the RSA [38] and, for the first time, its homomorphic property is described in [39].

Over the years, several Homomorphic Encryption Schemes were designed and realized in order to make even more sophisticated, secure and efficient this new approach. The survey [40], starting from RSA, describes all the most important Homomorphic Encryption algorithms designed. It shows the advantages and the disadvantages of each algorithm presented, in particular, referring to its efficient implementation. In the Cloud era, the Homomorphic approach is the solution for a concrete problem about data security and end-user privacy. For this reason, many studies tried to find a solution exploiting the Homomorphic Encryption main features. For example, in [41] the authors try to explain the problem, and, in particular, it describes how Homomorphic Encryption can solve the problem presenting several use-cases and several approaches.

The research presented in [42] provides a solution through which manage the data in security is possible. In particular the paper provides a layer on MongoDB Server transparent for end-user. Even in this case the solution makes use of Homomorphic Encryption in order to perform some computation on encrypted data. The work is interesting and provide a real solution for a NoSQL Database like MongoDB, widely used in Cloud Computing but, unlike our solution, the architecture is not designed for providing Cloud Services. Indeed, the database is exposed directly to end-user and there is no SaaS services provided. A similar situation is presented in [43]. In this work, a searching scheme in privacy condition is realized and tested. The research exploits Homomorphic Encryption in order to carry out data research and query without having to decrypt the data stored. Even in this case the Database considered is MongoDB. The work is interesting and it could represent a solution for the encrypted data querying problem. The architecture is pretty simple and it is not

really innovative and, even in this case, unlike our work, the solution is not provided to the end-user according Cloud "rules".

An interesting work about Cloud Data management is carried out in [44]. Here, the authors have designed a system to share reserved data in security exploiting the Homomoprhic Encryption to realize the "long-term" privacy concept and to ensure a trusted data flow between an hypotetical data owner and an hypothetical data analyst warranting the privacy of data owner but also his correctness in the data sharing. On this issue, an evolution is performed in [45]. In this article, a real service for the end-user is provided. Indeed, here, a secure and efficient privacy-preserving solution for exploring genomic cohorts in a real operational scenario is described. The solution exploits Homomorphic Encryption to stores and computes genomic data avoiding, as described, some well-known attacks and threats. This solution is not grouped in Cloud Computing pool services but, implicitly, provides a service secure managed software for end-users that can be also deployed in in a not-trusted environment without affecting the privacy of users. As we have seen, the healthcare use-cases are very common in this kind of service, because, as we have already said, healthcare is one of the most sensitive fields in ICT Applications. The solution proposed in [46] faces the contacts-tracing problem, during the COVID-19 Pandemic, exploiting a Homomorphic Encryption Scheme to ensure the privacy of each person that uses the app designed. The operations that verify the contact between two patients are performed over encrypted data in a totally transparent way. Even in this case, the service is not provided according the Cloud approach. Moreover the solution that we propose is more general and applicable on more use-cases.

In the paper [47] we have another example of homomorphic encryption implemented within a healthcare case and developed to researchers and medics. In the situation presented, in particular, the data encrypted and elaborated are genethic data. This kind of data are elaborated in order to diagnostic particular diseases. The peculiarity of this implementation is the use of a Multi Key Homomorphic Encryption Scheme. In the architecture designed the genetich data are provided and exploited by several different hospitals. This kind of Homomorphic scheme allows the presence of more public key and more permissioned users enabled to perform computations on data or decrypt them.

A strong evolution is actuated in CryptDICE system [48]. The solution proposed in that work is innovative and, in contrast to the solutions we have mentioned above, it's collocated in a real Cloud Architecture. Indeed, the research in question, presents a flexible and generic data access system that runs on top of the main models related to Cloud Computing. It

represents a framework through which it's possible to manage the data in a advanced way allowing the secure storing and the secure computation. It exploits more than one Encryption Scheme including an Homomorphic one in order to implement several ways to reserve data according to the application level use-case. The architecture designed is sophisticated and we can consider it like a Platform as a Service example. It is a framework provided to developer to realize secure application according Cloud standards. This research is strictly related to our work even if we want to increase the abstraction providing a ready service for end-user. Indeed, the final consumer will be not necessarily a developer, but we imagine he will be a generic tenant that wants to stores and elaborate data. Moreover our architecture, that we'll discuss in Section 4.3, can be considered *plug and play* because it's deployable in any classic Cloud Infrastructure always ensuring the privacy of tenant.

4.3 HE-Enabled Cloud/Edge Architecture

The system architecture depicted in Figure 4.1 shows how our homomorphic engine works around two main layers: Edge layer for data management; and Cloud layer for processing and service provisioning.

4.3.1 Edge Layer

The Edge layer is in charge to manage the collected plaintext data. In this layer, we have the implementation of the Homomorphic Encryption Scheme. Each Edge device generates, at first boot, the Key Pair useful for the Encryption and the Decryption of the collected plaintext data sending the Public Key to the Cloud server in order to allow the Cloud Side computation. The Edge devices are considered trusted and authenticated for each request carried out towards the Cloud layer. The data will be collected at the Edge layer through the end-user interface or IoT components that can detect the data through integrated tools. In the architecture, the devices within the Edge layer are strictly related to the customer tenant that is involved in the service providing. They are secure devices that are managed and administrated only by it. This approach ensures that the plaintext data is never collected by a third-party Cloud entity. The Edge layer through secure communication interacts with the Cloud layer that stores the data already encrypted through the typical tools used in data management. Figure 4.4 describes the main tasks of layers considering a real possible tenant like a *hospital*. The Edge layer, as shown, hosts the data received by all possible cases inside the hospitals. It could receive data from IoT devices and healthcare wearable devices (even in

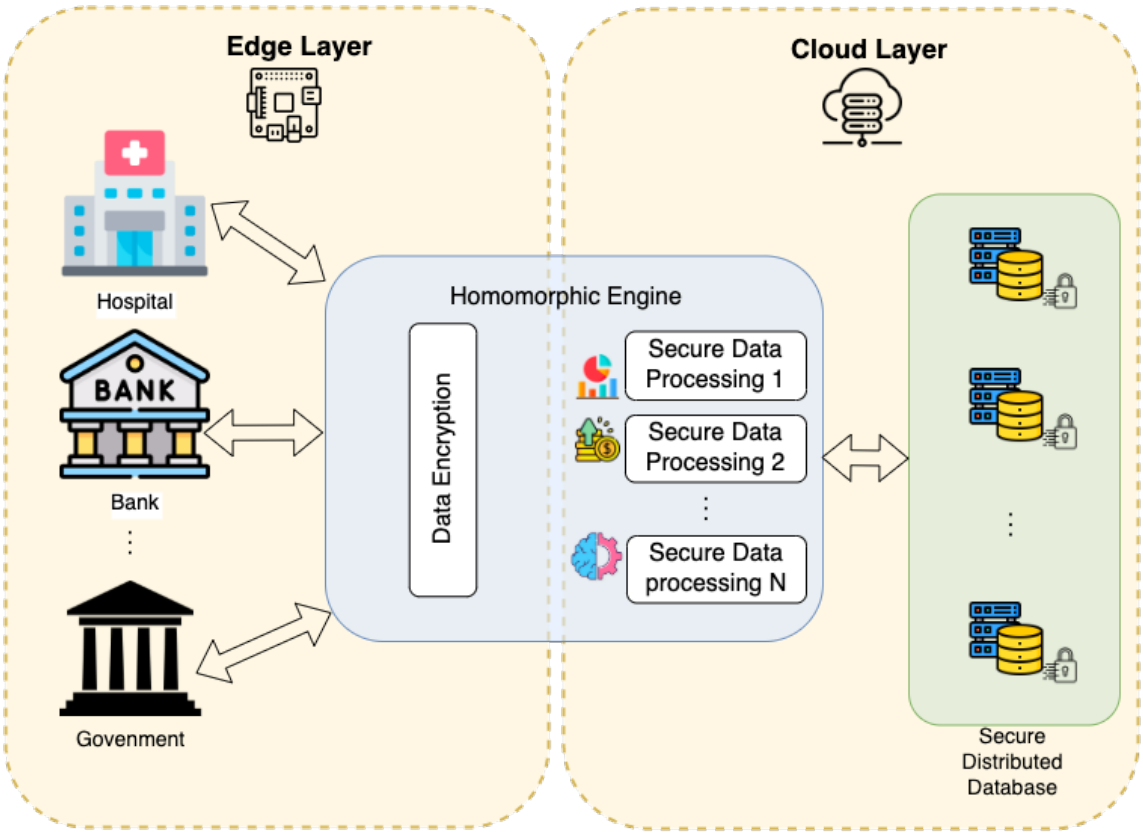

Figure 4.1: Homomorphic engine architecture.

tele-health scenario) that measure vital parameters of patients (e.g., heartbeat, blood pressure, oxygen saturation, etc.) but it could also receive data inserted from Doctors' reports and analysis, and also from the hospital administration that wants encrypt, stores and elaborate administrative data (e.g., therapies prices, revenue and so on). The Edge devices present in the tenant are responsible to acquire the data, encrypt and send them in a secure fashion to the Cloud layer and its position is not necessarily in the Hospital. The encryption will be actuated through the key pair generated at the first boot. After the key pair generation, but before the first data uploading, the tenant can retrieve the public key by the Edge layer. The key generation is dependent on the HE algorithm considered in the implementation phase which will be discussed in Section 4.4. Figure 4.3 describes, in detail, the components present within the considered layers. The Edge Layer hosts three main components:

- The Data Interface is the software interface that manages the data uploaded by the tenant. This component is a generic component that in the implementation phase can be developed to manage different human-readable formats.
- The Encryption Component encrypts the data updated through the Data Interface

Component. It exploits the key pair that has already been created. It abstracts the possible used encryption algorithm.

- The Client transmits the data over a network (e.g., Internet) to the server placed in the Cloud layer

The communication channel that connects these components is abstracted. For this reason, the components do not necessarily have to be inside the same physical devices. The components can be deployed in different devices always belonging to Edge layer.

4.3.2 Cloud Layer

The Cloud Layer includes a system architecture that manages the data involved without knowing its nature. The Cloud is totally agnostic about the state of the data. The same is valid for data processing.
The Cloud Layer functionalities in the proposed architecture are mainly two:

- The first is related to the collection and storage of the data transmitted by Edge layer.
- The second is the providing of actual service to the tenant.

The former functionality is accomplished thanks to the elements described in Figure 4.3. The latter are two:

- The modeler receives the data transmitted by the Client and, based on how the data received are structured (according to a specific format), it performs an insert task in the database.
- The database hosts the received data.

As Figure 4.2 shows the other main functionalities in which the Cloud Layer is involved include:

- The User Interface by which the tenant employees can access to the service. It could be accomplished in different ways. In the typical SaaS Cloud Model, the User Interface is a web user interface. This element allows the actual interaction with the service provided by the final user.
- The Homomorphic Engine represents the other side of the HE scheme. It is constituted by the operations that are possible to perform on the data. The architecture proposed by us does not indicate a priori the operations to be performed. The processing could

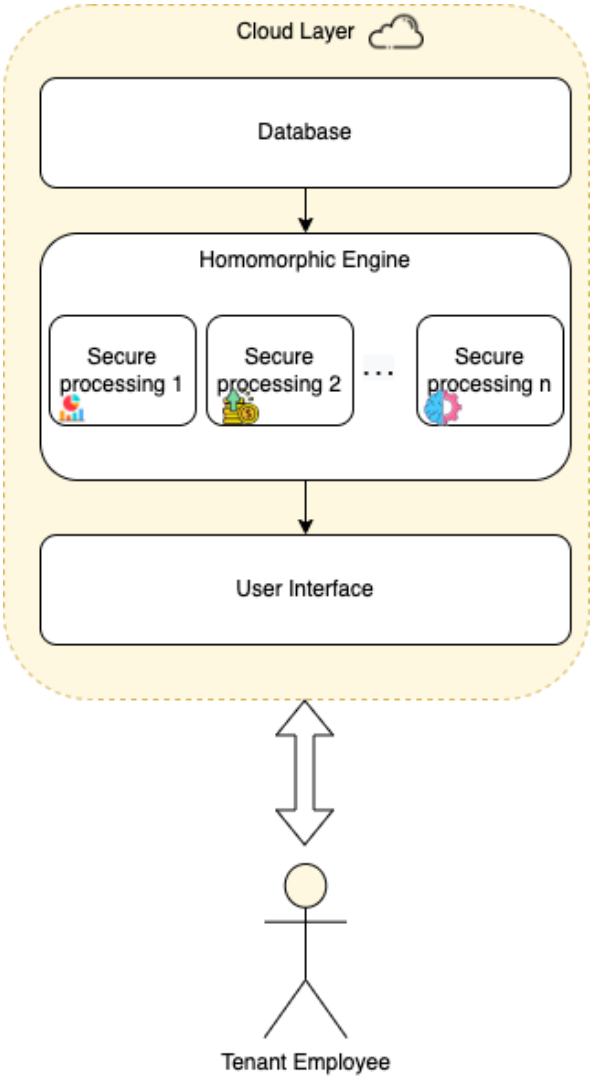

Figure 4.2: Cloud Layer detail service providing.

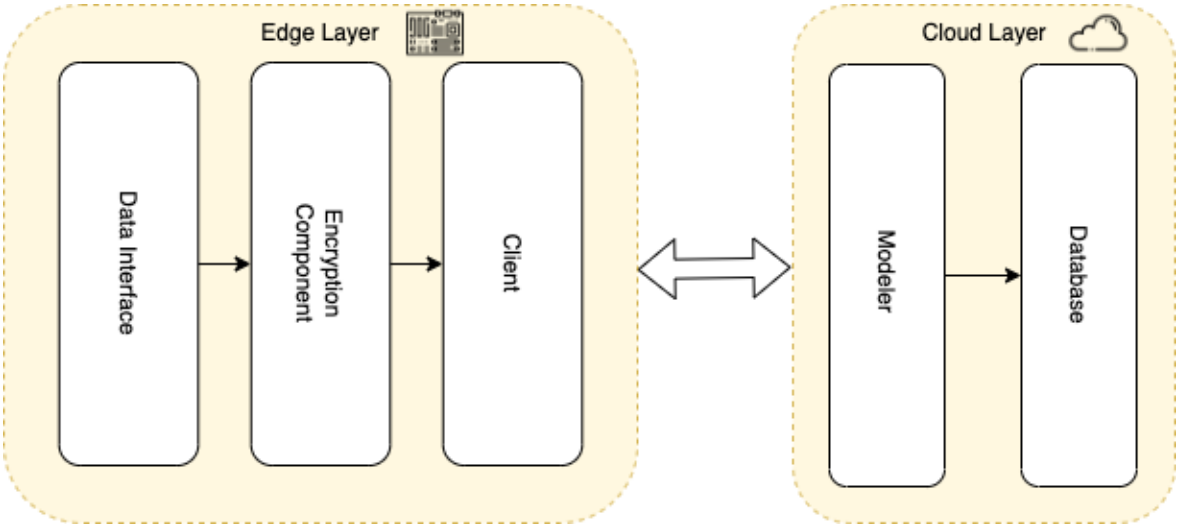

Figure 4.3: Layers detailed components in insert phase.

be different and based on what kind of elaboration the service wants to provide users. Each elaboration could be integrated next with respect to the first deployment. For example, they could be implemented through a serverless FaaS approach by which the provider can deploy different elaborations. Obviously, the main requirement is that each processing can communicate with the database that hosts the encrypted data.

These parts of the architecture do not hold the Private Key involved in the process, since it will never be able to decrypt the stored data. The communication between Cloud and Edge layers has to be secured, authenticated and encrypted. The detail of the security in communication channel and the management of accesses is out of the scope of the work described here.

From the point of view of the service provider, the process placed within the Cloud layer abstracts the Cloud mechanisms below. In summary, the final Software provided as a Service is, in turn, abstracted also for the Service provider.

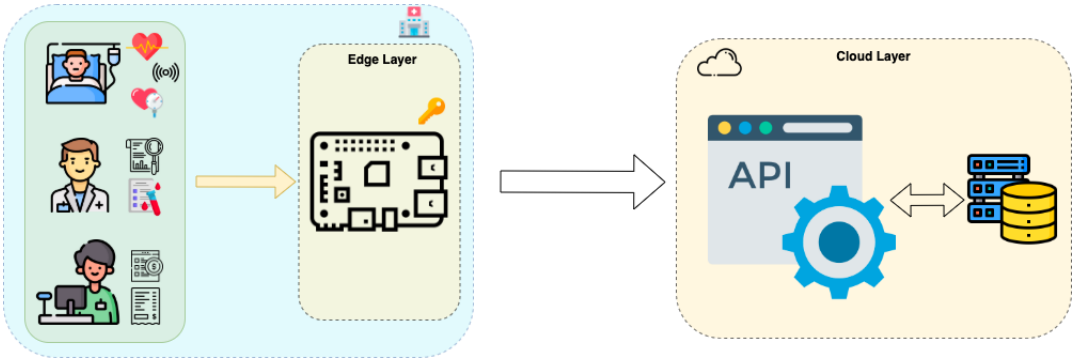

Figure 4.4: Use-case architecture detail.

4.4 Implementation

This Section describes a prototype implementation of the previously described architecture. In particular, its development is focused on the libraries used for Encryption functionalities. In our implementation, we want to exploit two different libraries and two different algorithms, both accomplished through the *Python3* programming language. They are the following:

- Python-Paillier [49, 15]. As the name suggests, it is a library that implements the Paillier algorithm in the Python language. It is one of the most famous HE schemes. It is a probabilistic algorithm based on a modular n-th residue problem. It is partially homomorphic and, in particular, it is homomorphic with respect to the sum operation.

- PyFhel [50] is another important Homomorphic Encryption library based on Microsoft SEAL Backend. It is a piece of Microsoft framework that implements HE at low level. The algorithms implemented are the BFV [36] algorithm for the operations on integer numbers and the CCKS one [37] for the operations over real numbers.

In our implementation, we imagine a real healthcare use case in which a medical facility or a medical doctor exploits the service provided in order to perform some statistical computations over encrypted data. For example, we figured out that the administrative employees of the medical facility update the data through a CSV file in order to store them in an encrypted way and to perform some operations over them. Another example could be a medical doctor that from a patient's home carry out some vital parameters measurements.

4.4.1 Edge Layer

In the Edge Layer, the implementation regards the first process involved in the flow which is the key pair generation and the process for the Encryption of the collected data. As we have described in the Section 4.3, the Edge layer hosts three components. The Data Interface for the data uploading, the Encryption and the Client. The first and the third ones, in this phase, are implemented through a Docker Container in which a Python Script is executed. The Data Interface was accomplished through the Flask Framework to expose a Web Server that can collect data. It takes the data uploaded and uniforms them like a JSON discoverable in the Encryption phase. The Client consists of a script in which an HTTP request towards the Cloud layer is carried out. As we have above mentioned, the main focus of our work is related to the HE scheme. For this reason, the Encryption Component was implemented through the two libraries above described.
First of all, as described, the Edge Layer is involved in the key pair generation implemented through the function provided by the libraries.

```
from Pyfhel import Pyfhel, PyPtxt, PyCtxt
HE = Pyfhel()
HE.contextGen(p=65537)

# Key Generation
HE.keyGen()

#Store KeyPair
HE.savepublicKey(pk_file)
```

```
10 HE.saveContext(contx_file)
```

Listing 4.1: PyFhel Key Generation

As we can see in the Listing 4.1, the key pair generation, with the PyFhel library, consists of the creation of a context to which the modulus is passed to. The creation of context allows for key pair generation. After that, they are stored in the filesystem in order to use them when necessary. Once the public key was generated, it is available for the tenant and it will be useful when it wants to get the result of computations over encrypted stored data.

```
1
2 from phe import paillier
3 import pickle
4
5 pub, priv = paillier.generate_paillier_keypair()
6 pickle.dump( pub, open( "pub.key", "wb" ) )
7 pickle.dump( priv, open( "pub.key", "wb" ) )
```

Listing 4.2: Python-Paillier Key Generation

The key pair generation in *python-paillier* library is almost the same. Even in this case, we store the keys in a filesystem in order to retrieve them when necessary. The Encryption operations performed by the Encryption component foresee the encryption of the data collected and uniformed by the Data Interface component. This component exploits the function provided by the libraries to encrypt the data that is sent to the Client component.

4.4.2 Cloud Layer

The implementation of Cloud layer was focused on the two functionalities. The first one is related to the insertion phase. This task is performed thanks to two elements: Modeller and Database. The Modeller is the component that takes the data transmitted by the Edge layer and starting from them, creates a model for the database storage. Moreover, the Modeller is in charge to perform the data insertion within the Database. This component is a Web interface developed through Flask framework. The Database component is implemented through the NoSQL DBMS MongoDB deployed in a Docker container. In our specific use-case, the considered model is organized in a single collection as shown in Listing 4.5.
The specific type of each field can change based on which library is used to encrypt the data in the insertion phase. Furthermore, the Cloud layer is connected to the final service provided to the tenant. In particular, it hosts all the components useful for the final service providing. As Figure 4.2 shows and as we have described in Section 4.3, the components involved in the

flow are three. The first is the Database that hosts the encrypted data. Its implementation has been already discussed. The communications between MongoDB and the other components of the architecture are managed through the python MongoDB driver *pymongo*. The second is the Homomorphism Engine that hosts all the homomorphic functionalities and the third is the User Interface that exposes the elaborations' result. The homomorphic processing carried out in our implementation are basically two:

- A cumulative sum on a set of filtered results.
- A cumulative average on a set of filtered results.

For each functionality, we must distinguish the two libraries used.
In the case of *python-paillier* library, we have our numeric data stored as an array of two elements. One element is the ciphertext and the other is the exponent used in the key pair generation phase. These parameters are then passed to the constructor in order to re-instantiate the EncryptedNumber object.

```
1
2 cur = db[collection].find(filters_applied)
3 sum = 0
4 for elem in cur:
5     ciphertext = int(element[homomorphic_field][0])
6     exponent = element[homomorphic_field][1]
7     encryptedelement = paillier.EncryptedNumber(public_key, ciphertext, exponent)
8     sum = sum + encryptedelement
9
10 return sum
```

Listing 4.3: Python-Paillier Cumulative Sum

In Listing 4.3 the first homomorphic processing is implemented using python-paillier library. As we can see, at the beginning, the data is retrieved from the database through a pymongo query. The field *homomorphic_field* represents the field on which the computation is carried out and it is passed by the user through the User Interface component. The same is for applied filters. They represent the filters, chosen by the user, applied to the inserted data. After the cursor is iterated and each EncryptedNumber stored, the cumulative sum is carried out.

```
1 cur = db[collection].find(filters_applied)
2 sum = 0
3 for elem in cur:
```

```
    ciphertext = elem[homomorphic_field]
    elem_he = PyCtxt()
    elem_he.from_bytes(ciphertext, encoding='int')
    elem_dec = HE.decrypt(elem_he)
    somma = somma + elem_he
return sum
```

Listing 4.4: PyFhel Cumulative Sum

Similarly, Listing 4.4 shows the implementation of the first functionality via the PyFhel library. The flow is almost identical. The only difference is the ciphertext type. In this case, the Encrypted number is stored on dabatase as a bytes stream. Indeed, we get that bytes stream, encode it and retrieve the Encrypted PyFhel Object. At this point, we can sum all the elements queried and returns the final cumulative sum. The cumulative average was developed accordingly.

4.5 Experiments

In this section, we'll see how we have evaluated the performance of the designed architecture and of the related implementation. For the experiments, we have imagined a tele-health scenario, as depicted in Figure 4.5. The data are updated through the Edge Layer, as described in Section 4.3, that is placed in the patient's home. As we have explained, the data model is generated by the component *Modeller*. In our test settings the model of the data uploaded is the following:

```
{
  "ID_User": "Patient's ID" ,
  "First_Name": "Patient's Name",
  "Last_Name": "Patient's Last Name",
  "Sex": "Patient's Sex",
  "Date_Of_Born": "Patient's Born Date"
  "Medical_Records_ID": "Medical Records' ID",
  "Hospitalizations": "Number of minutes of patient's hospitalizations",
  "Medicare_Payments": "Total Cost of treatments"
}
```

Listing 4.5: Schema Data Updated

The number of documents updated are 100000. The Aggregated operations have been carried out on *Hospitalization* and *Medicare_Payments* fields. The Encryption functionalities, as we have described, are deployed both on the Edge side and on the Cloud side. In this

scenario, we have simulated Edge devices with a Raspberry PI 4 with RAM Memory of 4GB. The Cloud Layer is simulated through an OpenStack infrastructure deployed on a Physical Server. The Virtual Machine created on that infrastructure has the following characteristics: RAM Memory 16GB, CPU of 10 cores and it runs Ubuntu 18.04.
We have evaluated the performance both of the insert phase and of aggregated elaboration. As said, the first one involves the Edge Layer, the second one involves the Cloud Layer. As we have seen in Section 4.4, the aggregated operations implemented (sum and average) are divided in two asynchronous phases:

- Query to the Database
- Homomorphic Aggregate Operation

For this reason, we have evaluated how each phase influences the final response time. In the performance tests, in order to evaluate the system, we have evaluated the aggregated sum. We have considered different portions of the dataset. In particular, we have tested respectively the 25, the 50, the 75 and the 100 percent of the dataset.

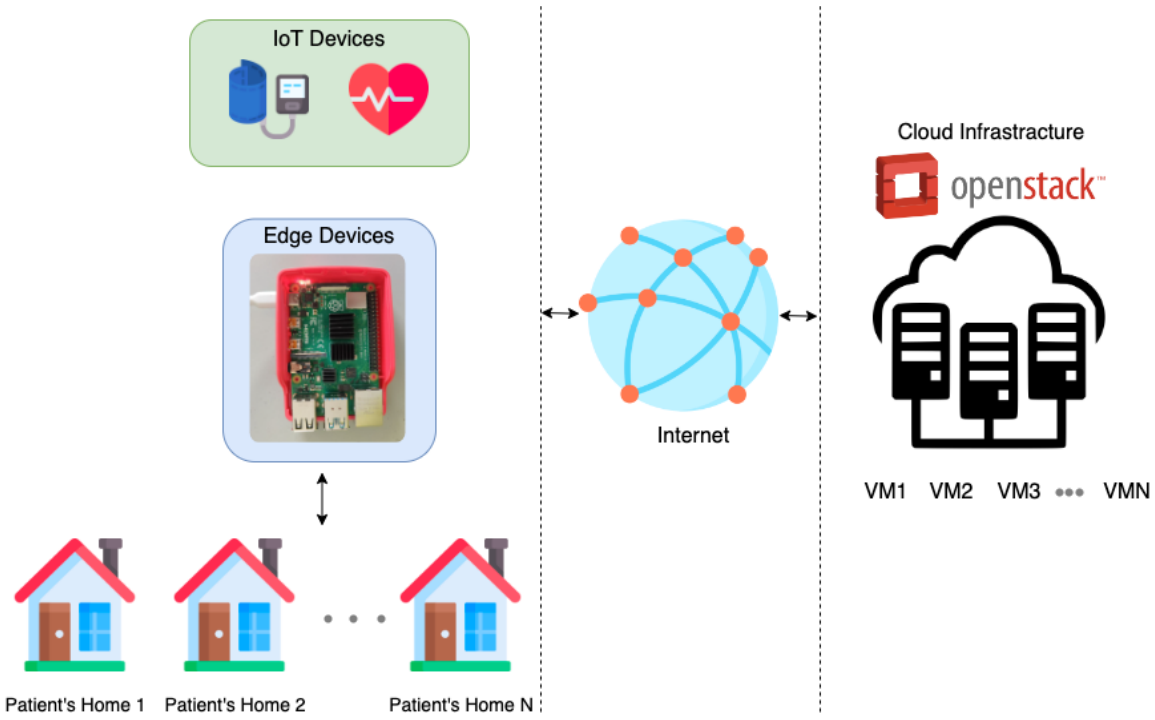

Figure 4.5: Experiment settings.

Moreover, we have also tested the insert time of the portions of the dataset considered in order to evaluate the average time for the insertion and the encryption of the data. Each test was performed 31 times in order to evaluate the performance considering the first query

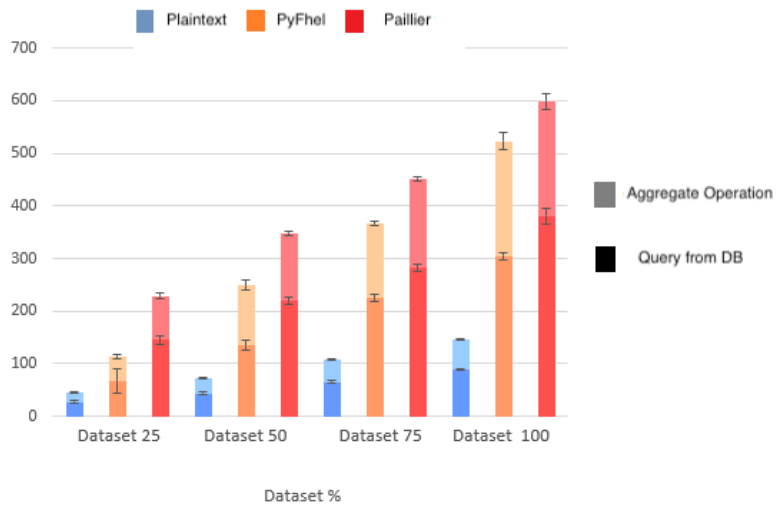

Figure 4.6: Time response of the system in Aggregation Sum operation.

response time affecting the whole dataset and the average query response time affecting the MongoDB Query Cache. This kind of evaluation is important because for a frequently changing data set the first query response time is very relevant, whereas for a rarely changing data set the average query response time of the following queries (on cached data) is very relevant. The sum test was performed also a single time in order to evaluate the query on not cached data. As we can see from the figures, in each case considered and evaluated, the PyFhel Libraries is more efficient with respect to the Pailler library. The reason is related to the implemented algorithm. The PyFhel, as we have already discussed, exploit BFV and CCKS Algorithm that are more sophisticated. Moreover, the considered library exploits the continuous support of the Microsoft framework, and for this reason, it could result in more updated and more maintained.

4.6 Remarks

The massive use of Cloud computing pool of services requires everyday better security in data management. Although Cloud Computing is a powerful and now indispensable tool, the security weakness is blinding. The work that we have proposed here can potentially solve one of the most important and studied security weaknesses of Cloud Computing. The solution proposed, with the help of Edge Computing in a Cloud-Edge continuum context, describes an accurate and applicable secure solution for data storing in a database and its elaboration.

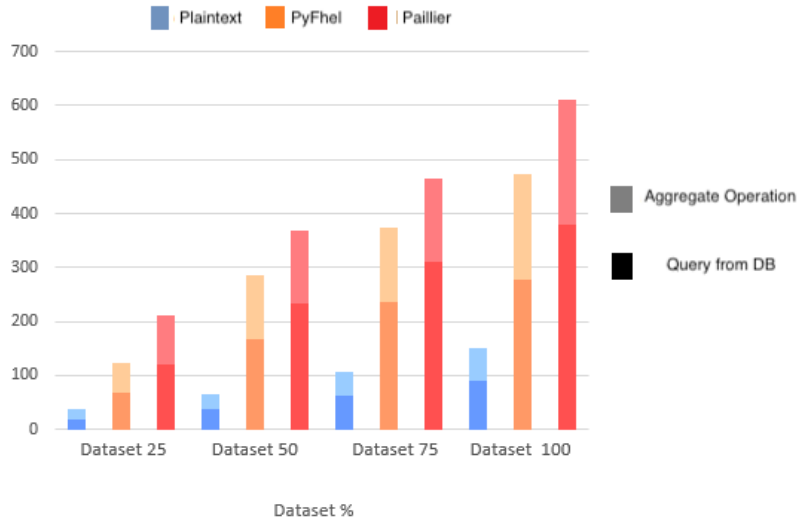

Figure 4.7: Time response of the system in Aggregation Sum operation - First Take.

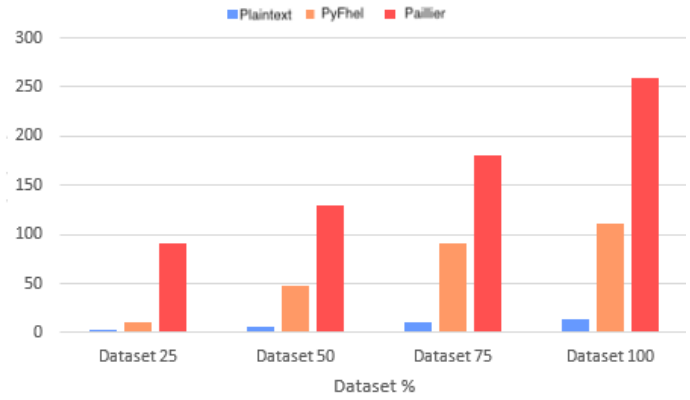

Figure 4.8: Insertion All Data Time response.

The tests carried out on the solution prove that the response times could result tolerable. In future work, we think that new algorithms and other interesting libraries could be implemented. Moreover, the architecture designed in this work does not bind all the homomorphic elaboration proposed here. We can imagine joining in this architecture and in the service provided even more sophisticated elaboration that, for example, involves Machine Learning operations on Encrypted Data.

Part II

Distribution of Cloud Computing Centralized Approach

CHAPTER 5

Cloud-Edge Workflows for Environmental Monitoring

5.1 Problem Statement

Cloud Edge Computing is an innovative paradigm in services computing field that has emerged as a pattern that combines the capabilities of cloud infrastructure with the agility and proximity of edge devices. This dynamic approach is revolutionizing the way we process and analyze data, offering an array of innovative possibilities across various domains. In particular, Cloud Edge Computing and associated workflows have become invaluable tools for addressing environmental challenges and achieving sustainability goals.

The environmental sector faces an ever-increasing volume of data generated by diverse sources, such as remote sensors, IoT devices, satellites, and drones. These sources provide crucial insights into climate, natural resource management, conservation efforts, and environmental monitoring. However, the sheer magnitude of environmental data presents unique computational and logistical challenges.

This is where Cloud Edge Computing comes into play. As we deep in 1, 2, Edge Computing allows to extend computing capabilities to the border of the network, closer to where data is generated, and as we saw in detail, this approach reduces latency, enhances real-time processing, and optimizes bandwidth usage. It allows for the seamless integration of cloud resources, edge devices, and workflows tailored to the specific needs of environmental applications.

The proposed work tries to exploit workflows in Cloud-Edge to perform an environment

forecasting leveraging Cloud and Edge infrastructure combined in order to overcome the weaknesses of the single infrastructure. To do that, two workflows were realized, in which the inference with a feed-forward neural network and a transformer is carried out.

The contribution of the work is the following:

- Design an Edge-Cloud architecture able to host two Workflows to make a distributed inference
- A Practical Implementation of the Workflows with Pegasus WMS

5.2 State of the art

During the last few years, several research works tried to combine Artificial Intelligence and distributed systems to optimize several factors in the whole computation and to exploit more distributed resources. The work proposed in [51] distributes Deep Neural Networks inference over distributed computing paradigms like Cloud, Edge, and end devices geographically distributed. The inference is distributed across all the paradigms used (Cloud, Edge, IoT) and an early-exit approach is exploited. It evaluates, in the inference phase, the correctness of the inference for each layer starting from the end device until the cloud. It can combine multiple end-layer devices. The solution described in [52] realized an inference distributed splitting the Layers of the Neural Network in a bootstrap phase and distribution according to specific layers. It proposed an architecture and a specific flow with the following tasks: Model Partitioning, Configuration, and Distributed Inference. The solution logically concatenates different nodes in order to spread the model partition and distribute the inference. In [53] the authors proposed a solution to split the Convolutional Neural Network (CNN) structure into smaller parts with the aim of reducing memory usage. The simulation results performed on VGG16 and ResNet18 networks for the classification of CIFAR10 images demonstrated a reduction in both the quantity of memory consumption and the number of computational operations.

In [54], the authors proposed a distributed inference framework called EdgeFlow. The solution is designed to split the inference tasks among multiple Edge devices. The results showed that the proposed solution is able to reduce the inference time by 40.2% compared with other solutions.

In [55], the authors proposed a solution that not only distributes a Deep Neural Network (DNN) across multiple edge devices but also optimizes computation resources and memory.

The solution has been validated by simulation of six edge devices running YOLOv2 DNN.

In [56], the authors proposed a Spark cluster for distributing the inference in TensorFlow. In particular, they compared the performances of four different configurations: i) a single Raspberry Pi 4B; ii) a cluster composed of two Raspberry Pi 4B; iii) a cluster composed of two Virtual Machines (VMs); and iv) a desktop computer. To validate the proposed solution they tested InceptionV3 on CIFAR10 and ImageNetV2. However, due to errors, all Spark-based clusters failed to perform the execution.

In [57], the authors proposed a solution to move the Machine Learning (ML) inference into IoT devices. The idea is to create a hierarchical structure that has accuracy and prediction comparable with a cloud-based solution. The experimental results showed that the accuracy of the hierarchical-based solution is the same as the cloud-based but the energy use and latency can be reduced by up to 63% and 67% respectively.

Although all aforementioned works treated different aspects of the distribution of ML inference, none of these consider a framework that is able to distribute the workflow between Cloud and Edge resources.

5.3 Motivation and Use Case

The Cloud-Edge Continuum, as mentioned earlier, combines two essential paradigms for service provisioning and deployment. It encapsulates the convergence of cloud computing and edge computing, providing a comprehensive approach to data processing and computation. This innovative architecture is a pivotal response to the dynamic requirements of modern applications, where low latency, real-time processing, and efficient resource utilization are paramount. Thanks to this new framework, we can harness the advantages of the cloud, such as computational resources, in combination with the benefits of edge computing, including low latency and enhanced security.

At its core, the Cloud-Edge Continuum bridges the gap between the vast computational capabilities of the cloud and the immediacy of edge devices. It seamlessly orchestrates data flow and processing tasks between the cloud and edge, enabling applications to leverage the strengths of both domains.

In our proposed solution, we introduce a Workflow concept within the Cloud-Edge Continuum environment to offload Artificial Intelligence (AI) inference tasks between cloud and edge infrastructure. This novel approach capitalizes on the low latency of edge computing for data collection and the robust computational capabilities of cloud infrastructure for

intensive inference tasks.

The adoption of a Cloud-Edge Workflow for AI inference in hydrogeological basin datasets is driven by the pressing need to address critical water resource management and environmental challenges. Hydrogeological basins, intricate systems of groundwater and surface water, play a pivotal role in sustaining ecosystems, providing drinking water, and supporting various industries. Consequently, the utilization of AI is essential for resource monitoring, environmental management, and disaster prevention.

In our proposed solution, we exploit a specific dataset related to our use case: the LAMAH dataset [58]. This dataset, covering the Danube River's basins across three different countries, offers diverse information, from geophysical data to historical water levels. Leveraging a Cloud-Edge Workflow enables scalable AI inference, facilitating the efficient extraction of intricate patterns and trends from this extensive dataset.

Furthermore, our solution emphasizes resource optimization, particularly in remote or resource-constrained areas. The distributed approach of the workflow maximizes resource efficiency, reducing energy consumption and costs associated to data transmission and cloud-based processing. The proposed solution performs AI inference in order to forecast one of the columns provided by the LAMAH dataset: the runoff of the basin expressed in the dataset as *Qobs*.

As of the time of writing, there is no existing solution that leverages the Cloud-Edge Continuum and Workflows for distributing AI inference processes. Through this approach, we aim to enhance the management and orchestration of AI inference across a variety of heterogeneous devices.

5.4 Design

In this section, the solution proposed will be described. In particular, its architecture will be deep and the workflow designed will be depicted and described. As mentioned, the proposed solution will create a Cloud-Edge Workflow for the inference of the water runoff of the baisins. LAMAH dataset provides different kinds of data for about 800 basins [58]. In our solution, after deep, a workflow for the forecasting of runoff in all basins is proposed. The work, in particular, as mentioned, will perform the inference of the label *Qobs* that describes the water runoff of each basin. Runoff forecasting is a regression problem that tries to foresee the amount of water flowing out by the basin on a specific day. Before the design of the architecture, some works related to the cleaning of the dataset and the analysis of data were

performed. Indeed, the dataset is very huge and several features belong to it. These operations were performed in order to carry out the training of the models exploited in the distributed inference.

5.4.1 Preliminary Works

The preliminary works performed before the design of the architecture were related to the model used for the inference of the data. In particular, two models are considered:

- A Forward Deep Neural Network with two hidden layer
- A Transformer model for the time-series data prediction

The models were trained by exploiting each basin dataset. Only 10 basins are considered for the training and 10 different models have been trained. Indeed, a different model for the basin was trained. This choice is related to the workflow design, as specified in the following sections, but also because a unique trained model should not be able to generalize the forecasting problem. As shown in Figure 5.1 and 5.2, Transfomer model reach a lower error on a test set after the training part.

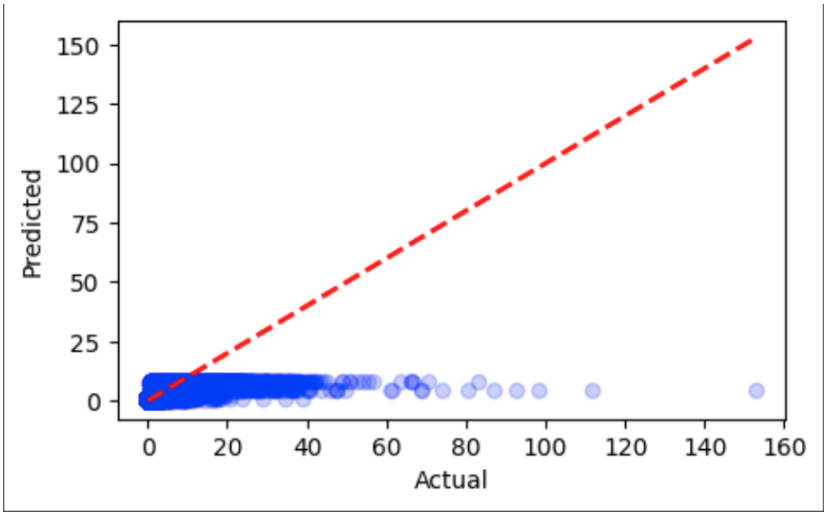

Figure 5.1: Error representation for feed-forward model

5.4.2 Architecture

The High-Level Architecture is described in Figure 5.3. As depicted by the image, two different levels are considered. The lower level is the Edge layer that interacts and collects the data victim of the inference. The *higher* layer is the Cloud Layer in which, obviously, the Cloud Infrastructure is hosted. Each Layer is part of the infrastructure

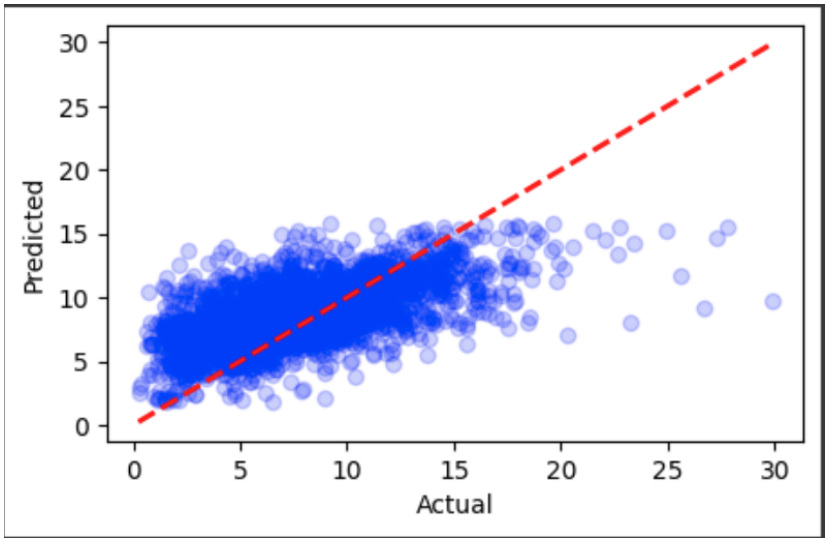

Figure 5.2: Error representation for Transformer model

5.4.3 Workflows Design

The designed workflows split the inference processes into different chunks that can be distributed among Cloud and Edge Infrastructures.

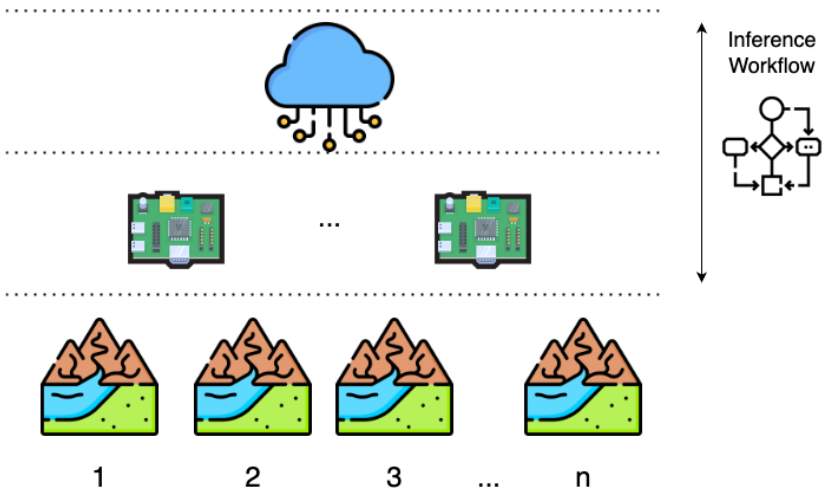

Figure 5.3: Solution Architecture

1. Workflows Objective: The primary objective of the two designed workflow is to distribute the inference process by dividing it into smaller partitions in order to offload that among Edge and Cloud.

2. Workflow Structure: The workflow's structure is tailored to partition the inference process. In particular, the workflow starts with inference data and retrieves the correct weights to load splitting the model into different parts. The Workflow's Jobs will be depicted in detail about both workflows.

Transformer Inference Workflow

The workflow is graphically depicted in Figure 5.4.
Job Descriptions:

a. Preprocessing (Job ID0000001): This initial job involves the conversion from the raw data collected to the input of the next jobs and the input of the models

b. Preinference (Job ID0000002): Although this job doesn't explicitly involve partitioning, it could be responsible for preparing the models and data structures required for the partitioned inference. In particular, during this job, the right weight set is loaded. Indeed, as specified, different weights are trained for different basins.

c. Generate Partitions (Job ID0000003): This job plays a pivotal role in the partitioning strategy. It takes the trained model from the previous jobs and generates multiple partitions of the model. These partitions will be exploited for the inference in the next jobs.

d. Inference Jobs (Job ID0000004 to ID0000007): Each of these jobs is an inference task, and it processes through different partitions of the model generated in the previous job. By splitting the inference into these separate jobs, it is possible to offload each job to a different infrastructure (e.g. Cloud or Edge)

e. Inference Decoder (Job ID0000008): This job performs the final inference that involves the Decoder of Transformer and that involves the Decoder Input and the Encoder Output

f. Conversion (Job ID0000009): The final job takes the processed data and converts it into a more usable format for further analysis or reporting.

Classical Feed Forward Network Inference Workflow

The workflow is graphically depicted in Figure 5.5.
Job Descriptions:

a. The jobs related to preprocessing, preinference and conversion of the final output and generating partitions have the same logic already described for Transformer Workflow.

b. Generate Partitions (Job ID0000003): Unlike transformer workflow, here, the Generate Partitions job splits the model into homogeneous partitions. Indeed, in *classical* feed-forward neural networks the layers are quite homogeneous among the model.

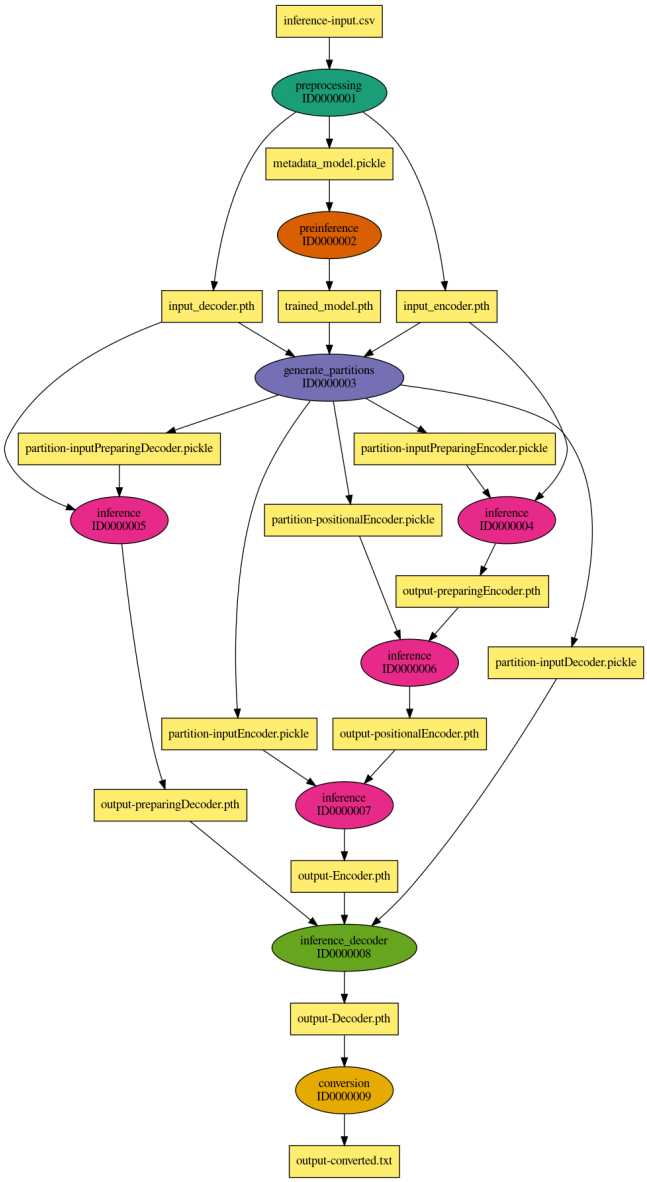

Figure 5.4: Workflow for Inference with Transformer model

5.5 Implementation

In this section, the practical implementations of the designed workflows will be described. The carried-out implementation has been focused on the workflows that, as already specified, will be deployed in a Cloud-Edge Continuum environment. The workflows have been realized through Pegasus WMS [59].

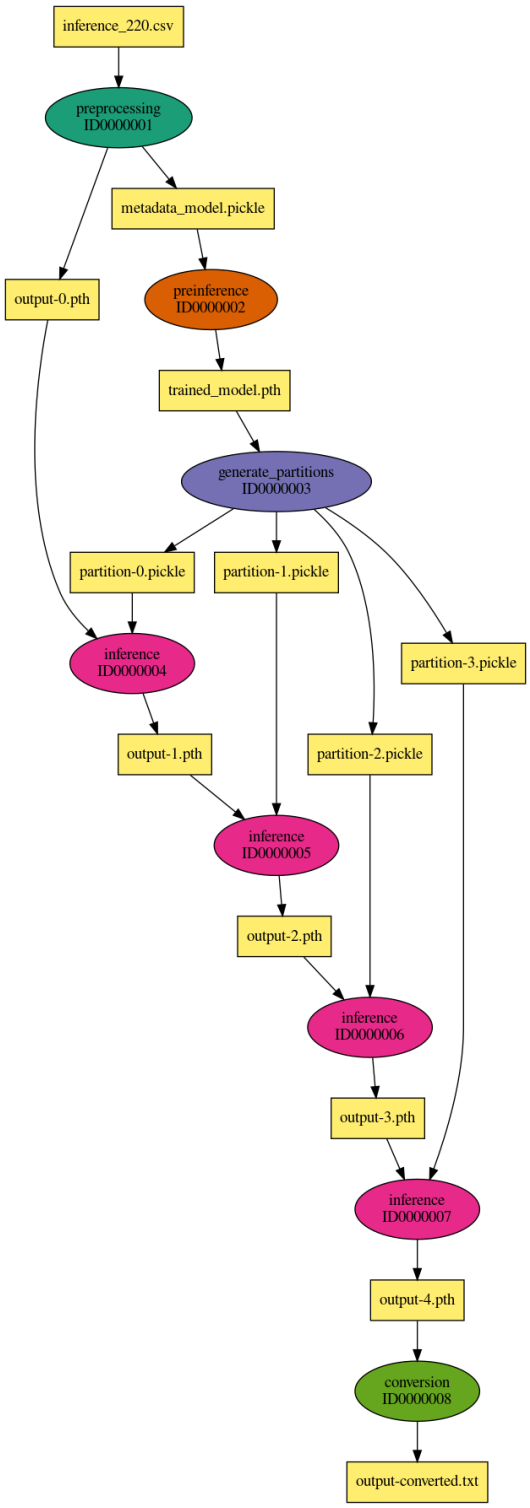

Figure 5.5: Workflow for Inference with Deep Feed Forward Neural Network

5.5.1 Pegasus

Pegasus Workflow Management System (WMS) is an open-source tool that empowers scientists to create custom abstract workflows using high-level APIs. For the execution of these workflows, Pegasus utilizes HTCondor, which is an open-source software framework

designed for distributed computing. HTCondor's primary function is to manage and schedule workloads across a cluster of computers or a grid of machines. In collaboration with HTCondor, Pegasus offers a range of functionalities that enable the transformation of an abstract workflow into a tangible representation in a .dag format. This transformation allows HTCondor's DAGMan to efficiently allocate resources and schedule jobs within the HTCondor cluster.

The HTCondor cluster foresees three different roles within a cluster:

1. Submit Node (Scheduler): The Submit Node serves as the entry point for users to submit their computational tasks to the HTCondor system. Users provide job descriptions, and the Submit Node interfaces with the HTCondor Central Manager to manage and schedule the job queue. It does not directly execute the tasks but takes charge of job submission and monitoring. To ensure seamless functionality, both Pegasus WMS and HTCondor binaries must be installed on this node, as Pegasus utilizes HTCondor utilities to present the final workflow.

2. Central Manager (Master): The Central Manager assumes a central role in the HTCondor cluster. It maintains an overview of the status of all machines in the cluster and oversees the matchmaking process. In the HTCondor system, each worker node periodically shares its characteristics with the central manager. The central manager gathers information from the scheduler and handles the task of matching jobs to available resources. It is instrumental in resource allocation and job distribution.

3. Execution Node (Worker): Execution Nodes represent the worker machines within the HTCondor cluster. These nodes are responsible for the actual execution of computational tasks scheduled by the Central Manager. Execution Nodes can have varying configurations, and they run user-submitted tasks in isolated environments, ensuring that tasks do not interfere with each other. HTCondor involves different architectures for the execution nodes (e.g. arm64, x8086, etc). This makes him a perfect tool to realize a Cloud-Edge environment.

5.5.2 Workflows implementation.

Pegasus exploits its own YAML structure to define the workflow. The YAML workflow file Here, the YAML used to implement in Pegasus the Workflows described in Section 5.4 will be shown and briefly described.

Listing 5.1: Feed Forward Neural Network Worfklow YAML Structure

```yaml
x-pegasus: {apiLang: python, createdBy: submituser, createdOn: '10-13-23T07:52:51Z'}
pegasus: 5.0.4
name: qobs_distributed_prediction
jobs:
- type: job
  name: preprocessing
  id: ID0000001
  arguments: [inference_220.csv, output-0.pth, metadata_model.pickle]
  uses:
  - {lfn: metadata_model.pickle, type: output, stageOut: false, registerReplica: false}
  - {lfn: inference_220.csv, type: input}
  - {lfn: output-0.pth, type: output, stageOut: false, registerReplica: false}
- type: job
  name: preinference
  id: ID0000002
  arguments: [metadata_model.pickle, trained_model.pth]
  uses:
  - {lfn: metadata_model.pickle, type: input}
  - {lfn: trained_model.pth, type: output, stageOut: false, registerReplica: false}
- type: job
  name: generate_partitions
  id: ID0000003
  arguments: [trained_model.pth, 4, metadata_model.pickle]
  uses:
  - {lfn: trained_model.pth, type: input}
  - {lfn: partition-2.pickle, type: output, stageOut: false, registerReplica: false}
  - {lfn: partition-0.pickle, type: output, stageOut: false, registerReplica: false}
  - {lfn: partition-3.pickle, type: output, stageOut: false, registerReplica: false}
  - {lfn: partition-1.pickle, type: output, stageOut: false, registerReplica: false}
- type: job
  name: inference
  id: ID0000004
  arguments: [output-0.pth, partition-0.pickle, output-1.pth]
  uses:
  - {lfn: output-1.pth, type: output, stageOut: false, registerReplica: false}
  - {lfn: partition-0.pickle, type: input}
  - {lfn: output-0.pth, type: input}
- type: job
  name: inference
  id: ID0000005
  arguments: [output-1.pth, partition-1.pickle, output-2.pth]
  uses:
  - {lfn: output-1.pth, type: input}
  - {lfn: partition-1.pickle, type: input}
  - {lfn: output-2.pth, type: output, stageOut: false, registerReplica: false}
- type: job
  name: inference
  id: ID0000006
  arguments: [output-2.pth, partition-2.pickle, output-3.pth]
  uses:
  - {lfn: partition-2.pickle, type: input}
  - {lfn: output-2.pth, type: input}
  - {lfn: output-3.pth, type: output, stageOut: false, registerReplica: false}
- type: job
  name: inference
  id: ID0000007
  arguments: [output-3.pth, partition-3.pickle, output-4.pth]
  uses:
  - {lfn: output-4.pth, type: output, stageOut: false, registerReplica: false}
  - {lfn: output-3.pth, type: input}
  - {lfn: partition-3.pickle, type: input}
- type: job
  name: conversion
```

```
  id: ID0000008
  arguments: [output-4.pth, output-converted.txt]
  uses:
  - {lfn: output-converted.txt, type: output, stageOut: true, registerReplica: false}
  - {lfn: output-4.pth, type: input}
jobDependencies:
- id: ID0000001
  children: [ID0000004, ID0000002]
- id: ID0000002
  children: [ID0000003]
- id: ID0000003
  children: [ID0000005, ID0000004, ID0000007, ID0000006]
- id: ID0000004
  children: [ID0000005]
- id: ID0000005
  children: [ID0000006]
- id: ID0000006
  children: [ID0000007]
- id: ID0000007
  children: [ID0000008]
```

Listing 5.2: Transformer Workflow YAML Structure

```
x-pegasus: {apiLang: python, createdBy: submituser, createdOn: '10-13-23T09:05:32Z'}
pegasus: 5.0.4
name: qobs_distributed_prediction_timeseries
jobs:
- type: job
  name: preprocessing
  id: ID0000001
  arguments: [inference-input.csv, input_encoder.pth, input_decoder.pth, metadata_model.pickle]
  uses:
  - {lfn: metadata_model.pickle, type: output, stageOut: true, registerReplica: false}
  - {lfn: input_encoder.pth, type: output, stageOut: true, registerReplica: false}
  - {lfn: inference-input.csv, type: input}
  - {lfn: input_decoder.pth, type: output, stageOut: true, registerReplica: false}
- type: job
  name: preinference
  id: ID0000002
  arguments: [metadata_model.pickle, trained_model.pth]
  uses:
  - {lfn: metadata_model.pickle, type: input}
  - {lfn: trained_model.pth, type: output, stageOut: false, registerReplica: false}
- type: job
  name: generate_partitions
  id: ID0000003
  arguments: [input_encoder.pth, input_decoder.pth, trained_model.pth]
  uses:
  - {lfn:  partition-positEnc.pickle, type: output, stageOut: true, registerReplica: false}
  - {lfn: input_decoder.pth, type: input}
  - {lfn: partition-inputEnc.pickle, type: output, stageOut: true, registerReplica: false}
  - {lfn: partition-inputPrepEnc.pickle, type: output, stageOut: true, registerReplica: false}
  - {lfn: input_encoder.pth, type: input}
  - {lfn: partition-inputDec.pickle, type: output, stageOut: true, registerReplica: false}
  - {lfn: trained_model.pth, type: input}
  - {lfn: partition-inputPrepDec.pickle, type: output, stageOut: true, registerReplica: false}
- type: job
  name: inference
  id: ID0000004
  arguments: [input_encoder.pth, partition-inputPrepEnc.pickle, output-preparingEncoder.pth]
  uses:
  - {lfn: input_encoder.pth, type: input}
  - {lfn: output-preparingEncoder.pth, type: output, stageOut: true, registerReplica: false}
```

```
  - {lfn: partition-inputPrepEnc.pickle, type: input}
- type: job
  name: inference
  id: ID0000005
  arguments: [input_decoder.pth, partition-inputPrepDec.pickle, output-preparingDecoder.pth]
  uses:
  - {lfn: output-preparingDecoder.pth, type: output, stageOut: true, registerReplica: false}
  - {lfn: input_decoder.pth, type: input}
  - {lfn: partition-inputPrepDec.pickle, type: input}
- type: job
  name: inference
  id: ID0000006
  arguments: [output-preparingEncoder.pth,  partition-positEnc.pickle, output-posEnc]
  uses:
  - {lfn: output-posEnc, type: output, stageOut: true, registerReplica: false}
  - {lfn: output-preparingEncoder.pth, type: input}
  - {lfn:  partition-positEnc.pickle, type: input}
- type: job
  name: inference
  id: ID0000007
  arguments: [output-posEnc, partition-inputEnc.pickle, output-Encoder.pth]
  uses:
  - {lfn: output-posEnc, type: input}
  - {lfn: partition-inputEnc.pickle, type: input}
  - {lfn: output-Encoder.pth, type: output, stageOut: true, registerReplica: false}
- type: job
  name: inference_decoder
  id: ID0000008
  arguments: [output-Encoder.pth, output-preparingDecoder.pth, partition-inputDec.pickle,
    output-Decoder.pth]
  uses:
  - {lfn: partition-inputDec.pickle, type: input}
  - {lfn: output-Decoder.pth, type: output, stageOut: false, registerReplica: false}
  - {lfn: output-Encoder.pth, type: input}
  - {lfn: output-preparingDecoder.pth, type: input}
- type: job
  name: conversion
  id: ID0000009
  arguments: [output-Decoder.pth, output-converted.txt]
  uses:
  - {lfn: output-converted.txt, type: output, stageOut: true, registerReplica: false}
  - {lfn: output-Decoder.pth, type: input}
jobDependencies:
- id: ID0000001
  children: [ID0000003, ID0000005, ID0000004, ID0000002]
- id: ID0000002
  children: [ID0000003]
- id: ID0000003
  children: [ID0000006, ID0000004, ID0000005, ID0000008, ID0000007]
- id: ID0000004
  children: [ID0000006]
- id: ID0000005
  children: [ID0000008]
- id: ID0000006
  children: [ID0000007]
- id: ID0000007
  children: [ID0000008]
- id: ID0000008
  children: [ID0000009]
```

The structure of the YAMLs provided in Listing 5.2 are the following

1. **Metadata**:
    - `x-pegasus`: This section contains metadata about the workflow, including the creation language, creator, and timestamp.
2. **Pegasus Version**:
    - `pegasus: 5.0.4`: Specifies the version of Pegasus used for the workflow.
3. **Workflow Name**:
    - `name: qobs_distributed_prediction_timeseries`: The name of the workflow, indicating its purpose related to distributed time series prediction.
4. **Workflow Jobs**:
    - The workflow consists of several jobs, each with a type, name, ID, and a list of arguments. These jobs are organized in a sequence, serving various purposes.

    a. `Preprocessing (Job ID0000001)`: Prepares data and produces output for use in subsequent jobs.

    b. `Preinference (Job ID0000002)`: Preprocesses data in preparation for inference.

    c. `Generate Partitions (Job ID0000003)`: Creates multiple data partitions, likely for parallel processing.

    d. `Inference Jobs (Job ID0000004 to ID0000007)`: Perform inference on different data partitions.

    e. `Inference Decoder (Job ID0000008)`: May assemble results from previous inference jobs.

    f. `Conversion (Job ID0000009)`: Converts processed data into a more usable format.
5. **Job Dependencies**:
    - The workflow specifies dependencies between jobs to ensure the correct order and coordination of tasks.

5.6 Remarks and next steps

In this study, we have successfully designed two Cloud-Edge Workflows capable of processing raw data collected from Edge devices. These workflows enable distributed inference, allowing each job to be flexibly deployed on either the Edge or Cloud infrastructure, catering to various computing requirements.

Our work includes the practical implementation of these workflows, which was seamlessly managed using the Workflow Management Systems, Pegasus. This practical implementation demonstrates the feasibility and effectiveness of our approach in a real-world context.

Looking ahead, our research opens up avenues for future investigations. An experimental evaluation of the workflow with varying job distributions between the Cloud and Edge infrastructure can provide valuable insights into performance optimization. Additionally, the next steps could involve the development and implementation of new models, considering different datasets and use-cases. These extensions will not only enhance the versatility of the workflows but also address the evolving needs of Cloud-Edge computing applications.

Part III

Distribution and Security Intersected Issues

CHAPTER 6

Secure Two-Factor Authentication (2FA) on Cloud Computing

Nowadays, the increasing complexity of digital applications for social and business activities has required more and more advanced mechanisms to prove the identity of subjects like those based on the Two-Factor Authentication (2FA). Such an approach improves the typical authentication paradigm but it has still some weaknesses. Specifically, it has to deal with the disadvantages of a centralized architecture causing several security threats like denial of service (DoS) and man-in-the-middle (MITM). In fact, an attacker who succeeds in violating the central authentication server could be able to impersonate an authorized user or block the whole service. This work advances the state of art of 2FA solutions by proposing a decentralized Microservices and Blockchain Based One Time Password (MBB-OTP) protocol for security-enhanced authentication able to mitigate the aforementioned threats and to fit different application scenarios. Experiments prove the goodness of our MBB-OTP protocol considering both private and public Blockchain configurations.

6.1 Problem Statement

Nowadays, we live in an era where the digital identity (DI) plays a fundamental role to securely access various kinds of services in our social and working life also enabling non-repudiation [60]. More in general, a DI univocally represents an entity (e.g., a person, organization, application, or device) into a digital system that allows performing non-repudiable activities within a system.

Thus, it is essential to use advanced security tools and methods aimed at guaranteeing the traceability of each action and the integrity of the data accessed. A consolidated approach to manage the DI legitimating applicaiton user's access is the Two-Factor Authentication (2FA) [61]. It requires two distinct forms of identification to authenticate the end user requesting to access an application and it typically involves the generation of a One-Time Password (OTP) in addition to the end user's credentials usually used to access the system. However, the OTP generation is a thorny task that could compromise the security of the whole system. One of the most notorious examples of OTP generation algorithm is the Time-based OTP (TOTP) [62], which is based on the Timestamp value of the request and on a shared secret that is generally stored in a centralized server managed by a trusted authority. The latter is limited in terms of security, as it can be vulnerable, among others, to man-in-the-middle (MITM) and denial of service (DoS) attacks. In fact, an attacker who succeeds in violating the central server could be able to generate an OTP and impersonate an authenticated user.

The main purpose of the worked shown in this chapter is to approach the generation and distribution of OTPs in a decentralized fashion to reduce the dependence from any Single Point of Failure (SPoF) and make the DI management system more robust against possible attacks. Furthermore, we considered the advantages offered by the Blockchain technology to develop a distributed 2FA protocol. Indeed, Blockchain is a consolidated solution for ensuring the integrity of data in a distributed network. In our view, Blockchain can be successfully adopted to replace the traditional central trusted authority, that is legally responsible for the OTP distribution, with a cooperative trusted environment that implements the same functionalities.

Following the above key concepts, we propose a decentralized Microservices and Blockchain based One Time Password (MBB-OTP) protocol for security enhanced authentication. In particular, we present a decentralized 2FA system that generates OTPs manipulating different pieces of information (i.e., user identity, request timestamp, and a shared secret). By adopting the microservices architectural concept each microservice hosts only a portion of the "secret" and execute sub-tasks of the whole OTP generation process.

The advantage of MBB-OTP is twofold: i) it is not exposed to MITM attacks: by splitting the 2FA service into different and independent microservices, in case of violation of one of them, the attacker would get totally decontextualized information without affecting the end-user security; ii) it can potentially mitigate DoS attacks using the microservices architecture principles: a compromised microservice can be easily replaced by a non-corrupted one.

The major contributions of this chapter are the following:

- we present an analysis of existing OTP solutions;
- we discuss the vulnerabilities of the classic authentication and OTP generation systems;
- we propose the MBB-OTP protocol , that is designed as a decentralized solution that makes use of Blockchain.
- we provide a preliminary assessment of the MBB-OPT protocol, analyzing the impact of Blockchain and highlighting the required trade-off between security, efficiency and costs.

The remainder of this chapter is organized as follows. A brief overview of the most recent OTP generation techniques and their security vulnerabilities is provided in Section 6.2. Starting from the traditional 2FA architecture and the TOTP algorithm, the design of the MBB-OTP is discussed in Section 6.3. An prototype implementation is discussed in Section 6.4. Experiments assessing the MBB-OTP in both private and public Blockchain implementations are discussed in Section 6.5. Section 6.6 concludes the chapter.

6.2 Related Work

In recent years, the evolution of digital services has brought to the development of more and more advanced mechanisms to prove the identity of subjects. Currently, there is no one-fit-for-all authentication protocol and its choice depends on the requirements of each specific application. This section highlights the state of the art, limitations and drawbacks of the most used approaches.

An important research that discovers all main solutions for OTP management including a secure web authentication with mobile phones, a piece of framework for securing sensitive input and a user authentication protocol resistant to password stealing and password reuse attack is described in [63].

Some researches have proven some weaknesses of 2FA centralized management, such as MITM, DoS and consequently masquerade attacks. Thus, several works have tried to improve the security in the 2FA implementations. Mainly, they tried to solve the main weakness present in the generation and transmission of the OTP.

To achieve a faster and secure OTP generation for Internet of Things (IoT) devices (including smartphones) authentication, different cryptographic algorithms have been proposed. In particular, AES cryptography has been adopted to encrypt the seed [64], formed by username and timestamp, and Elliptic Curve Cryptography and the isogeny property have been

analyzed [65] to improve the generation of OTP and IoT device authentication avoiding the use of the same keys for different communications. However, performance evaluations are not considered so the value of the research can not be fully appreciated.

The evolution of 2FA is the Multi-Factor Authentication (MFA) in which researchers enriched the typical 2FA with an external IoT device to complete the process [66]. Although this approach is valuable, because the transmission of double OTP is carried out through two different protocols and different technologies increasing security, it is unpractical in many cases as each user has to bring an external IoT device to complete the authentication. For the reasons above described, some researchers have focused their attention on the abstraction of the OTP lifecycle in order to provide multi-tenant OTP Cloud services with the same architecture. For example, in [67] a One-Time Password-as-a-Service (OTPaaS) solution is proposed. Although the idea is innovative, it is not verified how security is improved.

In [68] a 2FA violation is described. In particular, a network violation is analysed, highlighting how the 2FA violation is possible as a result of the hijacking of software installed on the client machine for OTP generation. This would not be possible with our implementation because the request is managed in a decentralized manner, protected by an Identity Provider. As mentioned above, the request of a new OTP is an authenticated action and, thanks to the distributed nature of the architecture, it is generated on the server in response to a user autentication request. The weakness of the transmission of OTP through Short Message Service (SMS) has also been analyzed and it was proved to be susceptible to MITM attacks. In this regard, solutions to reduce the risk of MITM attack through a micro-services architecture is discussed in [69] and [70].

This approach helps to drastically improve the security of 2FA protocols, and we progressed the research further by implementing a decentralized architecture, the micro-services paradigm and Blockchain technology.

Several solutions have been proposed in recent years to integrate the intrinsic features of Blockchain of decentralization, data non-repudiability and certification along with 2FA, but all comes with some limitation or compromise.

A 2FA-based system leveraging Ethereum to validate the user according to the possession of a Private Key, whilst the Public part is stored on the server, has been analyzed in [71]. This solution can be impractical in most the scenarios when a user wants to access a system from a mobile device and he/she has to carry the Private key in an insecure manner. Moreover, this solution does not involve the actual generation of the OTP in the Blockchain, but it simply stores a secret key. A 2FA solution implemented using the Message Queuing Telemetry

Transport (MQTT) protocol and Blockchain is discussed in [72]. In particular, the OTP is verified thanks to the Ethereum Blockchain and the Public key. However, such a solution is valuable, it is pretty theoretical and it is not verified experimentally. Furthermore, it implies that each user is bound with an Ethereum address and involved in almost a transaction. Another implementation of the 2FA implementing Blockchain technology is proposed [73]. Here, the user sends a key as a transaction to the server using a public Blockchain such as Ethereum, whereas, the server, having verified the key, generates an OTP encrypted with the user's public key. Only the user will be able to decrypt the message to obtain the OTP and can access the system. Although such a solution increases security in OTP generation, it forces each user to interact directly with the Blockchain. In [74], authors proposed OTP generation mechanisms without a trusted entity with the use of private Blockchain. This approach is interesting but the Blockchain utilization is limited to the OTP verification and it is not involved in OTP generation.

Differently from the aforementioned solutions, our MBB-OTP protocol is user-friendly because it is transparent to the end user which has all the technical details abstracted from the application.

6.3 The MBB-OTP protocol

The typical centralized OTP generation approach presents several weaknesses. As shown in Figure 6.1, it groups all the steps in a single service exposing applications to MITM attacks. Moreover, if a hypothetical attacker violates the system, he/she could easily simulate the login of any user and his/her OTP generation.

Figure 6.1 identifies three conceptual phases and related processes in the OTP management:

- The Authentication process receives an access request from the end user and it manages all the authentication, authorization and accounting functionalities. It provides the input to the generation of the new OTP.
- The OTP Generator process runs the actual OTP generation algorithm. Starting from a shared secret, it generates the temporary password to access the application.
- The Sender process delivers the generated OTP to the end user.

Our MBB-OTP protocol applies the microservices architecture paradigm [75] in the whole OTP lifecycle using three loosely coupled fine-grained and lightweight services, each one

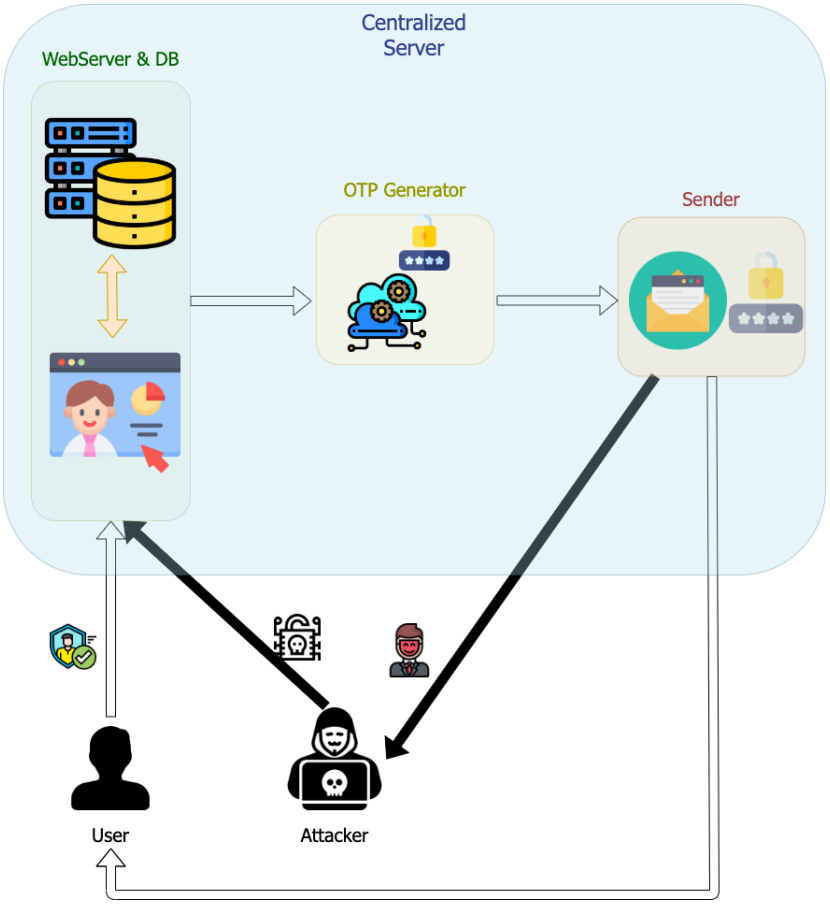

Figure 6.1: Centralized OTP management.

responsible for one of the three main phases identified in the aforementioned typical 2FA approach. Moreover, the MBB-OTP protocol involves the use of Blockchain to ensure the integrity of tasks performed by each microservice. Therefore, MBB-OTP abstracts any specific application which uses it: different applications could potentially use the same MBB-OTP implementation. In fact, as we can observe in Figure 6.2, the Application block is independent with respect to the other component blocks.

The MBB-OTP architecture mainly identifies the following component blocks:

- The Microservices: all the conceptual phases identified in the classical approach are implemented through different and independent microservices.
- Blockchain Gateway: it allows all microservices to write data about the tasks they perform in the Blockchain.
- Certification Authority: it manages the certificates that ensure secure communication between all the microservices.

Moreover, the protocol requires that the communication between the microservices has to be

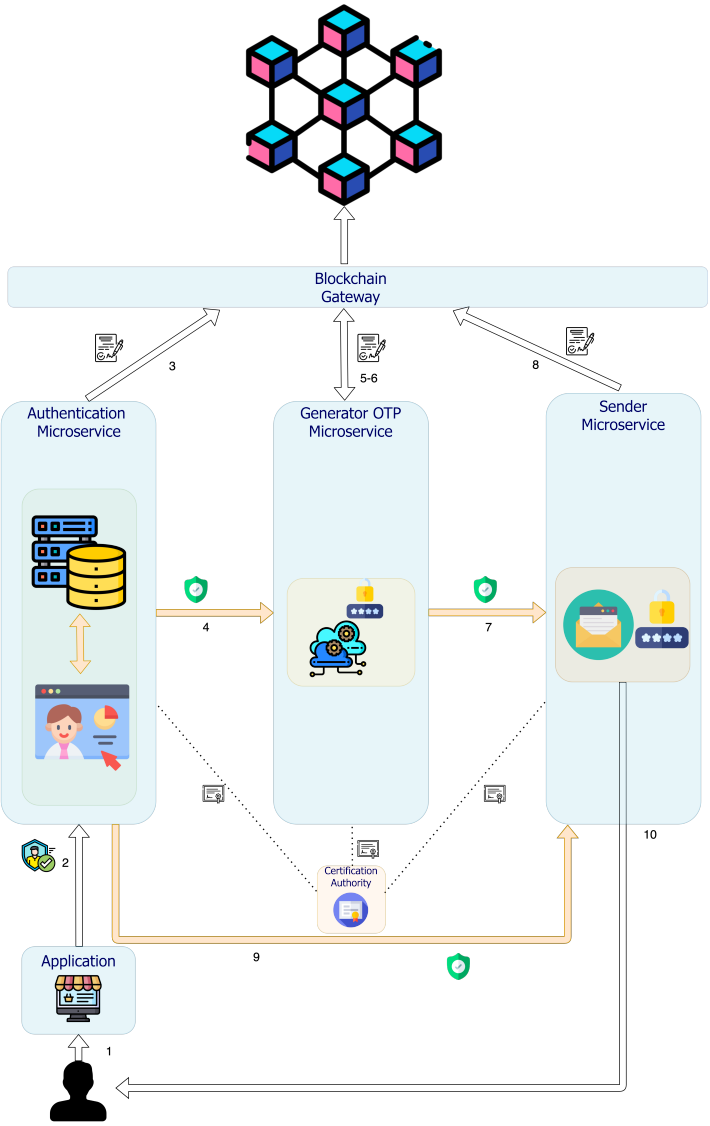

Figure 6.2: MBB-OTP protocol architecture.

secured. Indeed, a Public Key Infrastructure (PKI), through a Certification Authority manages all the certificates of the microservices in order to sign and encrypt the information exchanged between microservices. The Certification Authority releases, after the deployment of each microservices and during the configuration of the whole MBB-OTP system, a certificate ensuring that all communications are trusted. The integrity of each completed microservice task is ensured by the Blockchain network. In fact, the Blockchain plays an important role because it stores a shared secret that acts as a "seed" for the OTP generation. All the steps required by the MBB-OTP protocol to securely generate and deliver the OTP are shown in Figure 6.3:

1. The end user sends an access request to the application that requires an authentication with OTP.

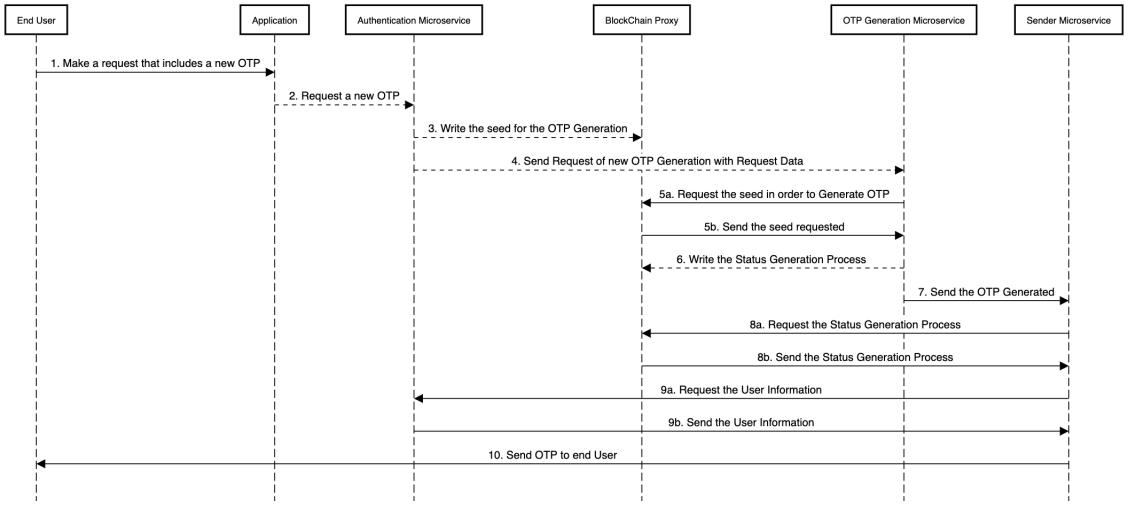

Figure 6.3: MBB-OTP protocol sequence diagram.

2. The application asks for the generation of a new OTP.

3. The Authentication Microservice (AMS) communicates with the Blockchain Proxy to write the seed for the OTP generation in the Blockchain. The information written in the Blockchain is signed with the Private Key of the AMS. The seed can be publicly available in the Blockchain because it is not a password on its own, and it needs to be combined with other elements to become an OTP. Furthermore, the seed is stored using the hash digest of a combination of user id and timestamp, so it is not possible to identify the seed of a requesting user without having the full set of attributes associated.

4. The AMS notifies the OTP Generation Microservice (OTP-GMS), providing the request timestamp and the key to retrieve the seed in the Blockchain. The information is encrypted and the communication is secure thanks to the PKI infrastructure.

5. The OTP-GMS retrieves the seed from the Blockchain through the information previously received from the AMS. After that, the OTP-GMS generates, using the received timestamp, the OTP with the TOTP algorithm, initializing it with the retrieved seed.

6. The OTP-GMS writes in the Blockchain the OTP generation process status to ensure the integrity of the flow.

7. The OTP-GMS sends the generated OTP to the Sender Microservice (SMS), encrypting the messing through the PKI infrastructure.

8. The SMS verifies the OTP Generation process status by retrieving the information from the Blockchain. This ensures that the OTP request is legitimately coming from the AMS

and it followed the entire flow as expected. In case the SMS is invoked without a valid Generation process status present in the Blockchain, the service would be refused.

9. The SMS requires to the AMS the User Information, such as email address or mobile phone number to deliver the newly generated OTP.

10. The SMS sends the OTP to the end user who can complete his/her authentication to access the application.

Once the end user submits the OTP to access the application, the latter verifies it. The verification process is analogous to the initial OTP generation steps. The AMS retrieves the OTP generation timestamp and the Blockchain access key value from the Database used for authentication. Then, it sends them to the OTP-GMS that gets the seed and along with the timestamp generates an OTP. In the end, it sends back the OTP to AMS that can verify the validity with the submitted one.

The MBB-OTP protocol drastically reduces the risk of identity violation. Unlike the centralized 2FA approach, it separates all phases, making each component independent. For example, the OTP-GMS simply takes the seed and the timestamp to generate the OTP, sending that to the SMS. It does not store and it does not know any information about the end user requesting the OTP.

6.4 Implementation

In this Section, we discuss a prototype implementation of our MBB-OTP protocol. Each microservice is implemented through the programming language Python 3, the Flask framework and the Gunicorn Python Web Server Gateway Interface (WSGI) HTTP server. Furthermore, each microservice is deployed within a Docker container to take the advantage of the virtualization technology allowing service portability, resiliency, and automatic updates that are typical of a distributed system acting either at the Cloud or the Edge [76]. The secure communication between the microservices is accomplished through the HTTPS protocol based on a certification authority created and managed through the OpenSSL tool. The implementation details of our prototype components are described below. The *AMS* component accomplishes Authentication, Authorization and Accounting (i.e., AAA functionalities). This microservice communicates with a DataBase Management System (DBMS) to retrieve registered user information. This is required to ensure that the entire process flow runs only if the requesting user is authenticated and recognized. The database implementation is out of the scope of the

work presented in this chapter and it can be carried out either with SQL, NoSQL, or NewSQL solutions. Moreover, the AMS communicates with the Blockchain Gateway through REST calls and writes a random seed on Blockchain as a key-value pair.The key of the Blockchain entry is the hash digest of concatenation of the user id and the timestamp. The value is the seed. Once the Blockchain transaction is committed, the AMS sends to the OTP-GMS, in the form of a REST call, the request timestamp and the hash digest of the Blockchain transaction by which the seed is stored. The pseudo-code about how this component works is provided in Algorithm 1. The *OTP-GMS* component leverages the TOTP algorithm using the seed stored in the Blockchain that is retrieved through a REST call to the Blockchain Gateway. After that, the component sends the OTP through a REST call to the SMS component. The pseudo-code about how this component works is provided in Algorithm 2. The *SMS* component is responsible to deliver the OTP to the end user requesting to access an application. This component can be configured with multiple communication protocols (e.g., SMTP, MQTT, mobile push notifications, and so on). The *Blockchain Gateway* component can be implemented either with Hyperledger Fabric Private Blockchain or Ethereum Public Blockchain according to the specific application scenario and security policy. In fact, some application might require the adoption of a private authentication service, others might tolerate the use of a public one. The first one uses Fabric 2.2 with an architecture based on four peer nodes distributed in four remote servers and four orderer nodes. The peer node receives updates and broadcasts them to the other peers in the network. The orderer node is responsible for consistent Ledger state across the network, it creates the block and delivers that to all the peers. The Public Blockchain implementation leverages the Infura web-based Application Program Interface (API) to interact with Ethereum. The Smart Contract is written in JavaScript (in the case of Hyperledger Fabric) and Solidity (in the case of Ethereum) programming languages, and it is designed to store the hash digest of the user id and the seed in the form of free text. The Smart Contract data structure includes two attributes: *Hash User Id* (String type), representing the anonymized user identification code; and *Seed* (String type), representing the value used as input for the TOTP generation.

6.5 Experiments

Blockchain represents an additional element in our MBB-OTP protocol with respect to a traditional 2FA one. For this reason, in this paper, we specifically focused on preliminary experiments aimed at assessing the overhead introduced by the private and public Blockchain

Algorithm 1 OTP Request

Class *Authenticator*
method *RequestOTP*(*username*)
  user_id ← *getAuthenticatedUser*(*username*)
  seed ← *generateSeed*()
  hashForSeed ← *generateHashForSeed*(*user_id*, *current_timestamp*())
  hashForLog ← *generateHashForLog*(*current_timestamp*(), *user_id*)
  saveSeedToBlockchain(*hashForSeed*, *seed*)
  saveLogToBlockchain(*hashForLog*, *msg*)
  sendToOtpGenerator(*hashForSeed*)

Algorithm 2 OTP Generator

Class *Generator*
method *GenerateOTP*(*hashForSeed*)
  seed ← *getSeedFromBlockchain*(*hashForSeed*)
  OTP ← *generateTOTP*(*seed*, *current_timestamp*())
  hashForLog ← *generateHashForLog*(*current_timestamp*(), *hashForSeed*)
  saveLogToBlockchain(*hashForLog*, *msg*)
  hashForLog ← *generateHashForLog*(*current_timestamp*(), *user_id*)
  sendOtpToSender(*hashForLog*, *OTP*)

in our protocol implementation. Specifically, we tested and compared the OTP generation time and resources usage in terms of CPU and inbound/outbound network traffic. With the private Blockchain approach, according to our MBB-OTP protocol the seed is stored in a private Blockchain by the AMS and retrieved by the OTP-GMS to generate the OTP. Such an approach obviously does not include any cost in terms of money. Whereas, with the public Blockchain approach, the seed is sent to a public Blockchain node to be added to the queue for immutable storage and mining. Time-to-mine and Ethereum (ETH) cryptocoin currency transaction cost are subject to Ethereum network traffic. In this case, the longer is the time required to mine, the higher is the cost to be paid for the transaction.

The testbed was arranged using a remote server with the following features: Intel® Xeon® E3-12xx v2 @ 2.7GHz, 2 core CPU, 4 GB RAM running Ubuntu Server 18.04. For private Blockchain, tests have been performed with 2 CLI, 4 peers and 4 orderers distributed across 4 different hosts. For public Blockchain, all tests have been performed using Ropsten Ethereum public Blockchain test network, leveraging 300+ available nodes with a real server load status. Tests have been executed for 100, 200 and 300 read and write operations, considering 95% confidence intervals and the average values. A summary of experiments setups is reported in Table 6.1.

Figure 6.4 shows the execution time difference expressed in seconds between the two system implementations to generate the OTP. On the x-axis, it is reported the two approaches

Table 6.1: Summary of experiments performed.

Parameter	Value
Contract address	0xa1032b39180538D7e17 23AedC39154C89A113BE5
Test executed for each scenario	[100, 200, 300]
Confidence interval	95%
Gas used by transaction	46,000
Gas price (Gwei)	200
Average cost per transaction (ETH)	0.0092326

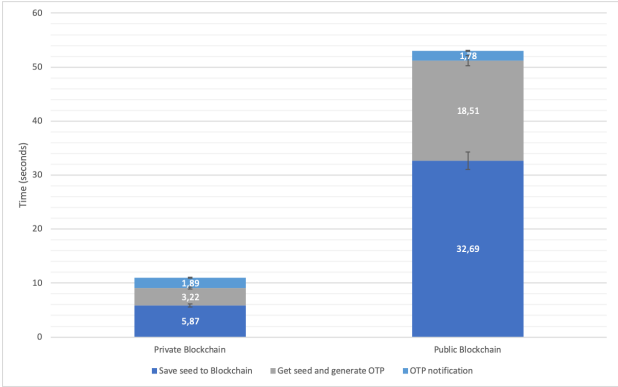

Figure 6.4: Execution time comparison for a single OTP generation (average) between private and public Blockchain.

for OTP generation. On the y-axis it is reported the processing time expressed in seconds. It is possible to observe that the time required to generate the OTP is five times greater with the Public Blockchain approach, due to the mining time required to store and retrieve the seed in the Public Blockchain. Instead, the average time required to notify the newly generated OTP is similar in both implementations.

Figure 6.5 shows the execution time difference expressed in seconds between the two system implementations to generate the OTP. On the x-axis, it is reported the differences to perform 100, 200 and 300 requests for OTP generation. On the y-axis it is reported the processing time expressed in seconds. From the graph, it is possible to observe that the time required to generate the OTP is largely different in the two implementations, and it shows a linear behavior. This is due to the mining time required to store the data in the public Blockchain that is considerably different from the private Blockchain approach.

Figure 6.7 shows the CPU usage % difference between the two system implementations for the OTP generation. On the x-axis, it is reported the differences to perform 100, 200 and 300 requests. On the y-axis it is reported the CPU usage %. It is possible to observe three different scenarios: a server that does not use the Blockchain, but it stores the seed in a local MySQL DBMS instance that is compared with both private and public Blockchain

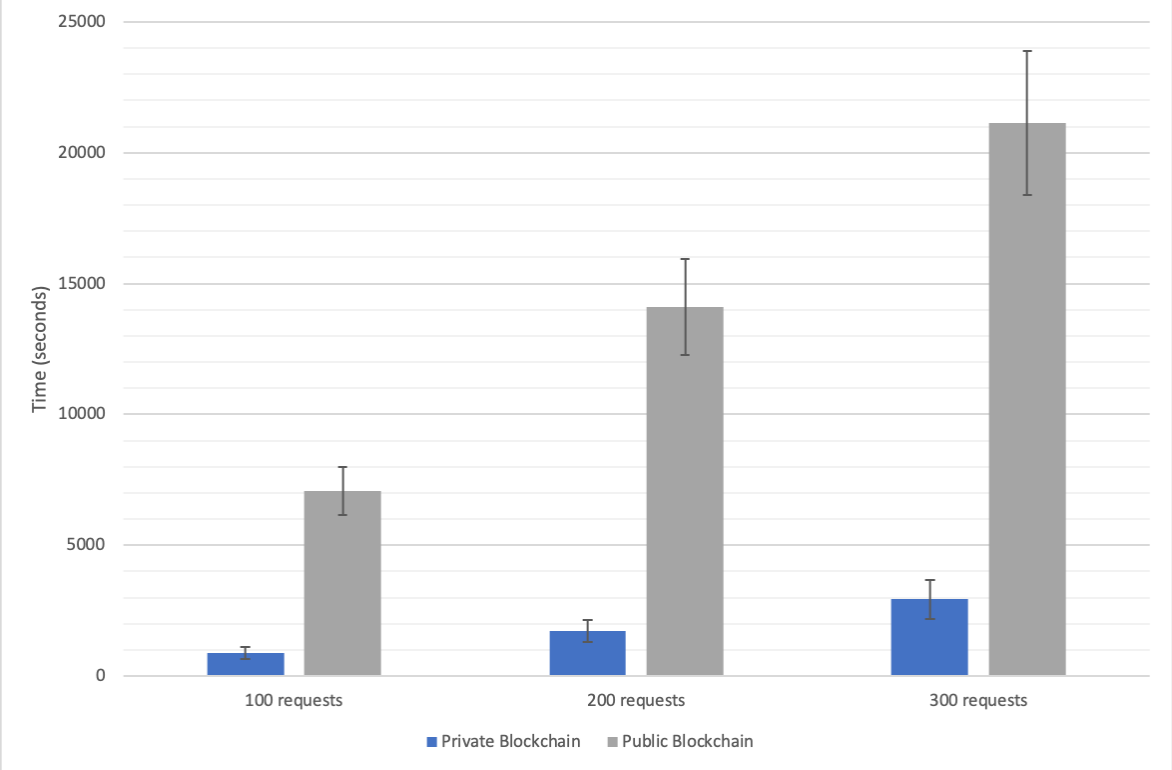

Figure 6.5: Execution time comparison between private and public Blockchain implementations for OTP generation.

implementations. It can be appreciated that the CPU usage % is kept in low range values and it is comparable in the three scenarios.

Figure 6.8 shows the inbound network difference for the three main Hyperledger Fabric components deployed in host 1. On the x-axis, it is reported the differences to perform 100, 200 and 300 requests, whereas on the y-axis it is reported the Inbound network consumption expressed in KB/s. Hereby, it is possible to observe that the network traffic grows up as the number of requests increases with a linear behavior.

Figure 6.9 shows the outbound network difference for the three main Hyperledger Fabric components deployed in host 1. On the x-axis, it is reported the differences to perform 100, 200 and 300 requests, whereas on the y-axis it is reported the Inbound network consumption expressed in KB/s. Hereby, we highlight that peer0.org1 has a considerable amount of outgoing network traffic. This is expected and is due to the block propagation required to share data with the other peers participating in the Private Blockchain network.

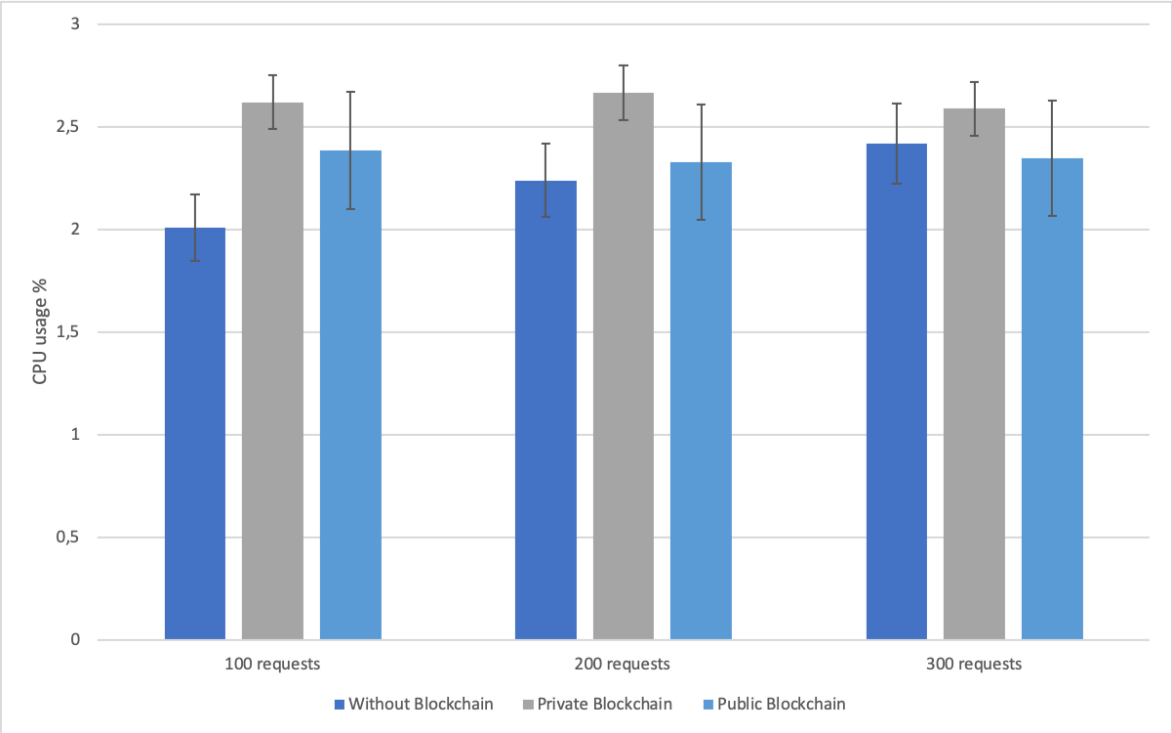

Figure 6.6

Figure 6.7: CPU usage % comparison between Private Blockchain and Public Blockchain approach implementations for OTP generation.

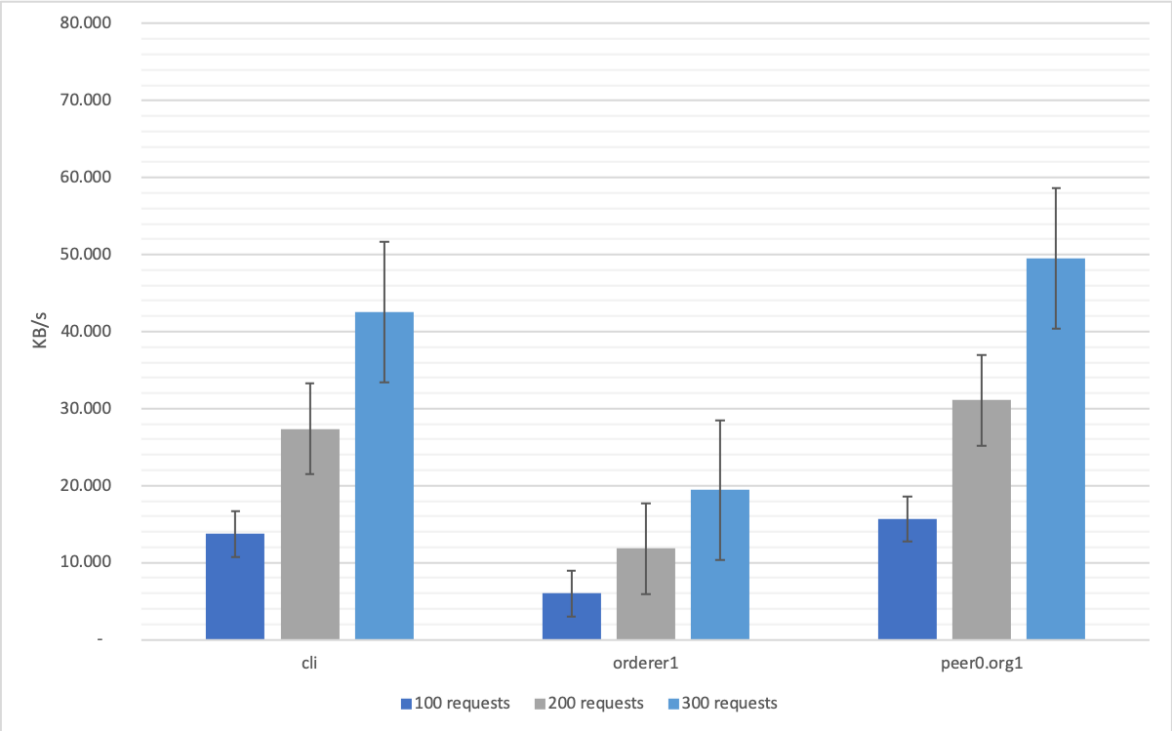

Figure 6.8: Inbound network comparison between Private Blockchain components required for the OTP generation.

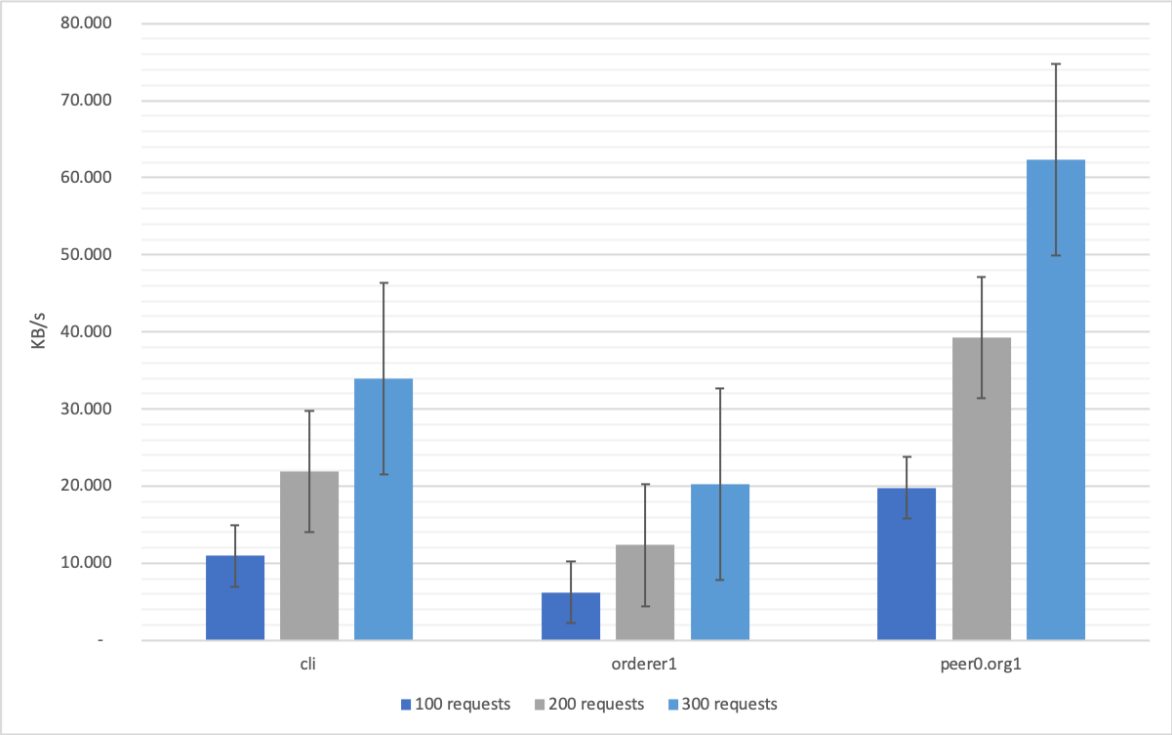

Figure 6.9: Outbound network comparison between Private Blockchain components required for the OTP generation.

6.6 Remarks

The increasing complexity of digital applications for social and business activities has required ever more advanced security mechanisms to prove the identity of subjects. T-OTP is one of the most common 2FA systems designed to achieve such an objective, but it based on a centralized architecture that represents a SPoF being consequently exposed to MITM attacks. To address such concerns, we proposed the MBB-OTP protocol that allows us to combine the security advantages of a decentralized 2FA system with the features of data non-repudiation and immutability that are typical of Smart Contracts. This protocol can be seamlessly implemented in any human-to-machine and machine-to-machine communication well-fitting the recent Cloud, Edge and IoT scenarios.

The solution has been prototyped and tested with both private and public Blockchain approaches. In the first case, although the trusted network needs to be built with a reasonable number of nodes to guarantee sufficient level availability and reliability, we observed low response times. Instead, in the second case, considering a high level of availability and reliability thanks to the larger number of nodes, we observed an acceptable degradation of response times.

Experimental results highlighted that the CPU usage % remains roughly unchanged

if using a DBMS such as MySQL to store the information, a private or public Blockchain, demonstrating that the MBB-OTP protocol can be implemented in any scenario without any performance degradation in terms of CPU usage. Obviously and as expected, the time required to store and retrieve any data to/from the public Blockchain heavily affects the overall performance, but waiting one minute to obtain a strongly secured generated OTP can be considered acceptable, considering the obvious security and robustness advantages.

The current MBB-OTP protocol, compared to traditional 2FA ones, mitigates MITM attacks and it can also potentially mitigate also DOS attacks by considering replicated microservices that can replace corrupted ones. In future works, we also plan to deepen such a mechanism, also considering intrusion detection systems. Furthermore, we plan to better investigate the performance of AMS, OTP-GMS and SMS considering different application scenarios.

6.7 Mobility Use-Case

The solution above mentioned has been implemented in a practical use case related to sustainable mobility. In particular, as described below, MBB-OTP protocol has been realized to manage a Bike long-rent service for Messina citizens. The physical authentication has been carried out through a 2FA that leverages the innovative protocol shown above. Sustainable mobility is a key objective for many Smart Cities. Hence the use case presents an application for sustainable mobility that aims at encouraging citizens to use low-impact vehicles instead of private cars. Through a partnership between the University of Messina and the Municipality of Messina city (Italy), we developed MuoviME, a digital application to assign citizens electric bikes, free of charge for a limited period of time. The key issue we addressed in developing such an application is security, both in terms of secure authentication of citizens that access the service and tracking of the whole assignment process, from the user's bicycle request to its restitution. To achieve this goal, we implemented a solution for the physical recognition of the user based on two-factor authentication (2FA) and Blockchain technology. The use case summarizes the secure-by-design approach, implementation details, and some experimental results on service efficiency.

6.7.1 Use case definition

Smart Cities have become a concrete reality all around the world. They enable innovative services for citizens to improve their quality of life, but also allow public administrations and private stakeholders to improve their processes and increase their business. Although

the advantages in promoting Smart City services and applications are evident, their actual implementation is limited by security issues [77]. In the last years, some researches tried to improve the security for Smart City digital environments. In particular, a huge number of Information and Communication Technology (ICT) platforms and tools were born to manage all main aspects of a Smart Cities like Access Control Management (ACM) and Authentication, Authorization, Accounting (AAA) policies. However, investigated results are hardly to be coupled with efficient or worthy applications, especially whenever these involve public administration and e-government processes [78].

In this use case, we try to overcome these issues presenting a new digital service that aims at pushing sustainable transportation in a Smart City by using electric bicycles instead of private and public vehicles powered by carbon-based fuel. Through a partnership with the Municipality of the Messina city (Italy), we developed a solution able to assign citizens electric bicycles, free of charge for a limited period of time. In particular, we implemented our solution addressing security issues both in terms of physical recognition of the user that accesses the service and for tracking of the whole assignment process, from the request of the user to have a bicycle to its restitution. To achieve such a goal, we adopted the 2FA (two-factor authentication) and the Blockchain technology integrated in the municipality processes for bicycle management. The Blockchain component ensures the integrity and the secure tracking of each step involved in the process [79], [80]. Specifically, this section discusses the design of the web platform to instantiate the services for requesting electric bike (e-bike), and for their assignment and access management. Furthermore, we provide implementation details, and some experimental results on the effective adoption of the developed solution in the Messina city use case.

6.7.2 MuoviMe Services Design

In this section, we present the design of MuoviME application for the long-time renting of electrical bikes to citizens in a Smart City. MuoviME comes from the requirements of the Italian project "Messina - A scuola e al lavoro con il Trasporto Pubblico Locale. Iniziative per promuovere la mobilita' sostenibile" that aims at promoting sustainable mobility in the city of Messina (Italy). The project includes among partners the University Messina and the Municipality of Messina. In particular, the project has the main purpose of discouraging the use of carbon-fuel powered vehicles and promoting activities that increase sustainable mobility within the city.

To achieve such a goal, we thought to a public web platform where each citizen can

submit a request to have, if available, an electrical bike assigned for free and for a limited period (e.g., max 90 days). Thus, we designed such a platform considering that the efficiency and the success of the offered service strongly depend on the overcoming of several security issues. Specifically, the platform must always be available and tolerant the failures that are typical of a centralized system, thence avoiding any possible bottleneck. Furthermore, the platform has to implement a strong authentication system able to guarantee the association between the digital identity of a citizen and the physical access to the assigned bike from the same citizen. In fact, at the time of delivery of the bike, the user, who will get it, must be authenticated, authorized, and recognized with the digital account that made the request. Moreover, the platform must guarantee that each phase of the bike rental process, is tracked by a third distributed system in order to ensure the whole process integrity. In our case, such a system is represented by Blockchain. In the following, we present the reference architecture we designed for the whole bike rental system workflow.

6.7.3 Architecture

The management of electric bikes in the MuoviMe application has been designed considering the benefits of a microservices-based approach [81]. In particular, the whole solution has been organized as a set of independent microservice components that interact each other and that can be deployed and dynamically managed taking the advantage virtualization in terms of resource optimization. Moreover, the microservices approach allows us to avoid the presence of a Single Point of Failure in the system, increasing the fault-tolerance of the service and its fast restore in case of attacks or technical issue [75].

The main microservice components of the MuoviMe architecture are shown in Figure 6.10. They include:

- The Web Server, that is globally reachable, through which each user can authenticate itself and access the MuoviME application.
- The OTP Management Module for the management of the 2FA needed by the user for the renting of the e-bike.
- The Blockchain Gateway allowing the communication with the Blockchain. It can abstract different kinds of Blockchain, i.e., either private or public.
- The Database Module ensures persistence data of Web Server and it interacts with OTP Management Module for the generation of OTP.

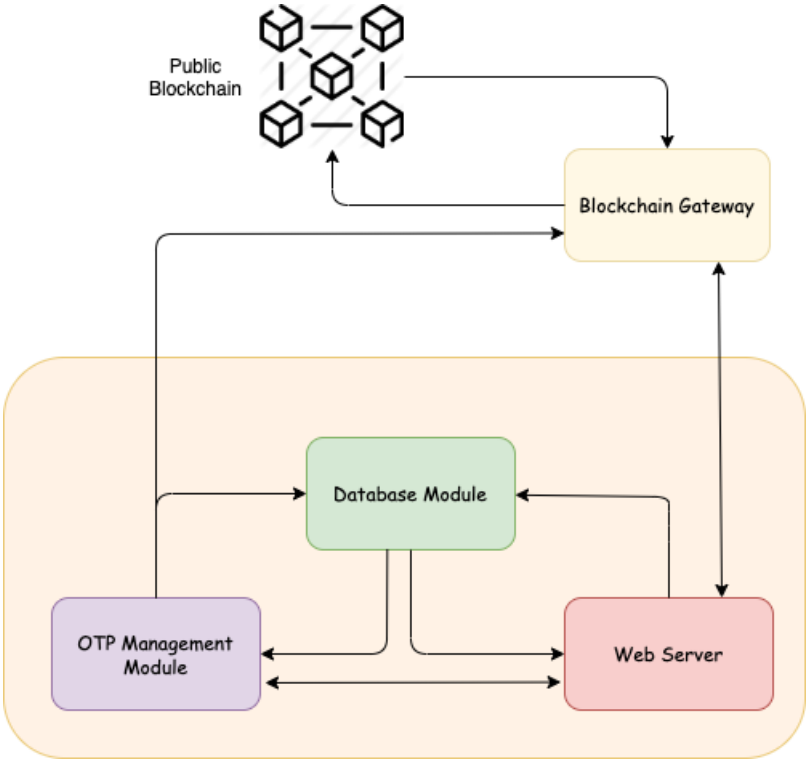

Figure 6.10: Reference architecture

6.7.4 Workflow

The actors involved in the MuoviME application are:

- The End User. The person who makes a new e-bike rental request. He/she interacts with the system in order to make a new request. This request is authenticated through the provided authentication platform.
- The Officer. He/she has the responsibility to validate or invalidate the e-bike rental request performed by end user. Only if the Officer validates the request, the MuoviMe workflow can go ahead.

Before the start the e-bike rental workflow, an initialization phase is required. Indeed, in this phase, the officer generates the key-pair useful for signing the validation or invalidation of each request. The generation has to be carried out only with the presence of the platform administrator. Indeed, once generated the key, the administrator signs the key generated in order to certify it. The workflow, from the initial request of the end user to the delivery of the bike is described in Figure 6.11 and consists of the following phases:

1. The End User makes a new request through the Web Platform. The request involves personal data of the User, identity document and so on.

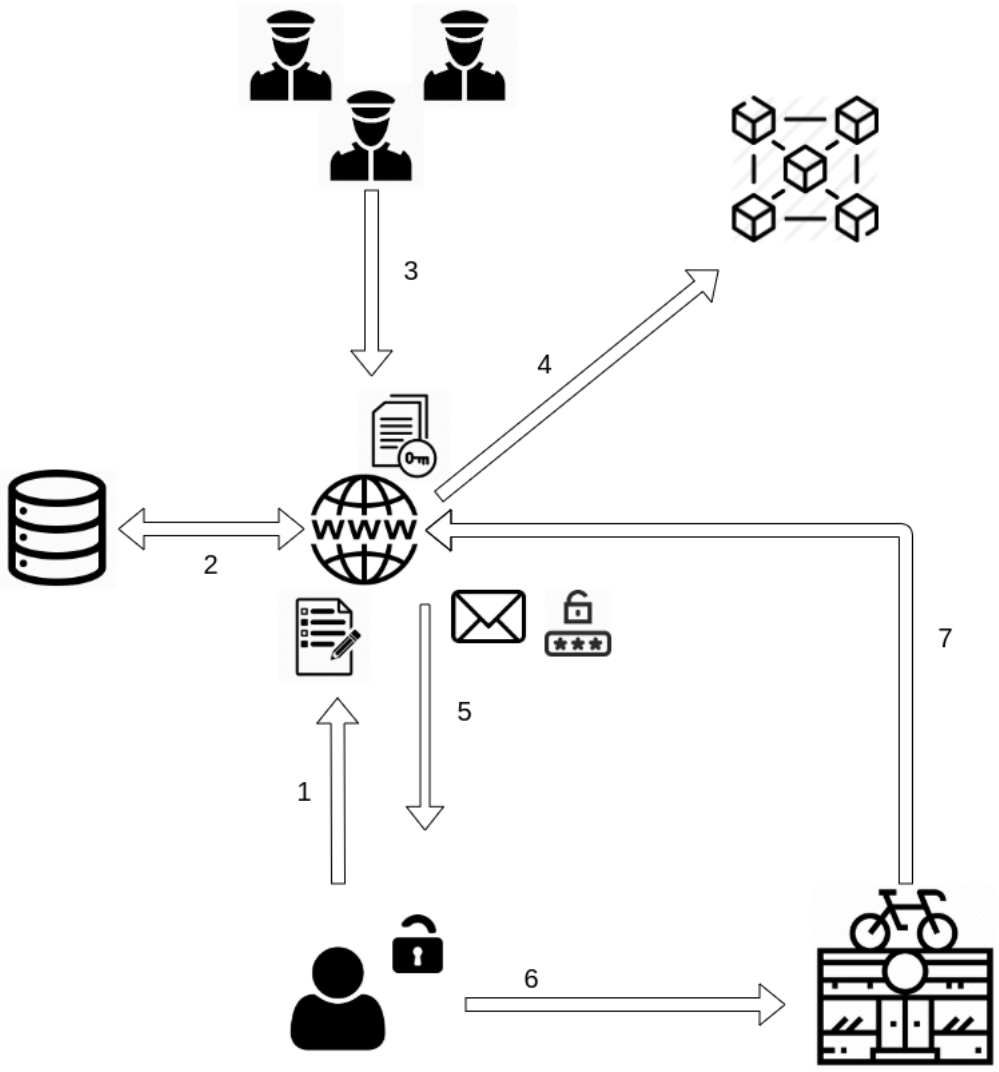

Figure 6.11: Use-case flow

2. The Web Server stores the request involving the end-user information inside Database.

3. The Officier checks the request submitted by the end-user and all the pieces of involved information: if the request is valid, he/she can validate the request. This step is not synchronous. The Officier can choice to validate more than one request in a single session. The request validation is signed with the private key generated in the initialization phase.

4. Once the request is validated, the platform, through the Blockchain Gateway (Figure 6.10), writes the information contained in the request in a Public Blockchain. In this phase, the platform makes a request to the OTP module requiring a new OTP for the delivery of the requested e-bike. The OTP generation is realised according to an innovative approach [1]. Indeed, the OTP module (Figure 6.10) is deployed in a

distributed fashion and it interacts with the Blockchain in the generation process. The OTP module is organized in microservices allowing to not be subject to Man in the Middle or DDoS attacks. Furthermore, the OTP module uses the Blockchain to store the seed used in the generation of the OTP for the user.

5. The End User receives the OTP through which he/she can collect his/her e-bike at the designated collection point.

6. The end user goes to the collection point to collect the requested e-bike. The system after the evaluation of OTP, through the exposed Web Server, records the delivery and all the information related to it.

Once the period of use is completed, the end-user has to give back the e-bicycle. In particular, he/she has to bring back it to the recollect point in order to make available again the bicycle used to new users.

6.7.5 Use case implementation

Microservices and web platform

As we said the architecture is focused and based on microservices. The implementation strategy is based on the Linux container concept [82]. In particular, the implementation of containers for each module considered in the architecture took place through the Docker container engine. Docker is a powerful tool widely supported by a community of developers. Indeed, the official registry containing all the images, always knows as "Docker hub", provides a huge number of images useful to deploy a various of services. In our case through Docker we have deployed the Web Server and the Database module. In particular, to develop the Web Application, we have used the Content Mangement System Drupal.
Drupal is characterized by a strong modulation. Moreover, it allows integrating custom modules, using PHP language, for the implementation of advanced functionalities. The Database module was implemented and deployed through the MySQL 5 Docker image. As said in Section 6.7.4 the Database module supports both Web Platform (e.g, Drupal Instance) and OTP Module.

End-User Functionalities

The main functionalities for end user were implemented through the available Drupal modules, whereas advanced functionalities were developed through an ad hoc custom module. For example, the Principal component of User Interface is provided by Drupal and by external modules supported by the community. Authentication and Authorization features are accomplished by native module of Drupal (core). The authentication is implemented through well-known and classical techniques. In particular, it could be implemented through a Lightweight Directory Access Protocol or relational database. Other important graphic functionalities were developed in Drupal by well-known external modules (e.g., web form). As said for the advanced safe functionalities a custom ad hoc module was developed. The latter implements several functionalities. First of all, the it integrates new tables in the Drupal default database that store important information about request, user and all workflow phases.

The integrated tables are three:

- *Submitted Requests* is the table that stores all pieces of request information useful for the validation and delivery of the e-bike.
- *Sellers Updates* keeps track of all digital warehouse.
- *Officier* stores the key-pairs of Officiers.

The first table is populated when the web form for the new request is submitted by an end-user. This mechanism is possible thanks to a powerful tool provided by Drupal, i.e., hook. This is an interface that allows a custom module to reuse both data and services provided by other installed modules. In this case a hook intercepts the submission web form and stores in the table some pieces of information related to the *Submitted Requests*. The second table is updated with the same mechanism. A specific web form is provided for the seller and each submission inserts a new row in the table *Sellers Updates*. The table is updated when a disclaimer delivering email has been send decreasing the number of submitted bikes. Another important hook that is used in our custom module is the "hook menu" of module *menu*. This hook allows creating a new path of the web site through a callback function that runs when the path is called in the browser. This hook allows integrating in the custom module several Representational State Transfer (REST) Application Program Interfaces (APIs) that can be called from the front-end. Moreover, the modules allow integrating javascript code. As we will discuss in the following, javascript code enables Asynchronous JavaScript And XML

(AJAX) calls allowing key-pair generation and validation.

KeyPair Generation

The KayPair Generation is implemented client-side through a javascript code embedded in our custom Drupal module using the AJAX approach. In particular the javascript PGP library is used to implement the geneartion of KeyPair with Elliptic Cryptography. The curve "ed25519" is used for the key-pair generation. The private key generated is ciphered with the Advanced Encryption Standard (AES) algorithm, using a password got by user-interface. After the creation and the encryption of private key an AJAX call communicates with the "officier/generation" endpoint exposed in the custom module developed through the "hook menu". The back-end stores the private key encrypted in the *Officier* Table.

Validation

The validation execution is the same of the key generation one. We have a javascript code that carries out a REST call thorugh AJAX, in an endpoint created with the "hook menu". The analogue callback function registers the validation or the invalidation storing the related data in the BlockChain transaction. In the validation the officier inserts through the user interface the password of KeyPair previously generated and signed by administrator, and decrypts the Private Key previously got by back-end and stored inside the database. With the private key the officier signs a JavaScript Object Notation (JSON) document in which the results of validation and the possible motivation are contained. The signature is stored in database through another exposed REST API.

Blockchain Communication

The Blockchain Gateway (Figure 6.10) is a container that acts as a proxy for communication with the adopted Blockchain system (either private or public). It is implemented through a python server built through the flask and guncorn frameworks and containerized, once again, through the Docker system. In particular for the implementation of private Blockchain Hyperledger Fabric was used whereas for the public Blockchain Ethereum Blockchain was considered. The Public Blockchain implementation leverages the Infura web-based Application Program Interface (API) [83] to interact with Ethereum. The Smart Contract is written in JavaScript (in the case of Hyperledger Fabric) and Solidity (in the case of Ethereum) programming languages. The Blockchain Gateway ensures integrity of Officier decision through a

Smart Contract that is structured to store hash digest of the officier id and the information about request validation.

6.7.6 Use case evaluations

In this Section, experiments are discussed. In the MuoviMe application, the Blockchain system could represent an overhead source that has to be carefully assessed since it plays an important role in the OTP generation process. In fact the response time introduced by Blockchain in the OTP Generation could negatively affect the user experience on the whole system. For this reason, experiments focused on the OTP Generation scaling up the system from the point of view of number of users who simultaneous send e-bike renting requests to the MuoviMe application. Specifically, we assessed the mean response time of the OTP module that, as explained, interacts with Blockchain system for the delivery of OTP. In particular, we compared the use of both private and public Blockchain systems. The difference in

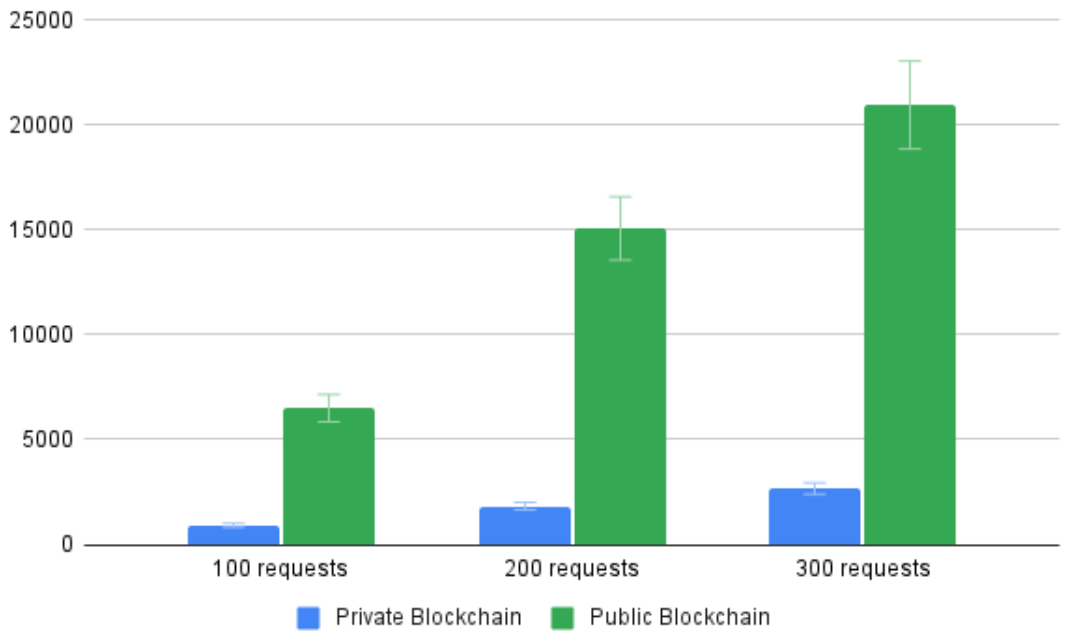

Figure 6.12: OTP time response.

execution time expressed in seconds between the two system implementations for generating the OTP is shown in Figure 6.12. The two approaches for OTP generation are shown on the x-axis. The processing time, expressed in seconds, is reported on the y-axis. We can observe that the public Blockchain approach takes five times longer to generate the OTP. This is due to the time required to mine the seed and obtain it from the public Blockchain system. Instead, both solutions take about the same amount of time to alert the freshly created OTP.

6.7.7 Use case remarks

In this section, MuoviME, a solution to promote sustainable mobility in the city of Messina (Italy), was presented, highlighting techniques for the physical recognition of users in the Smart City context. With our solution, the state of the art regarding access control by users in a Smart City has been advanced. Going into detail, the project, carried out in collaboration between the University of Messina and the the Municipality of Messina distributed several e-bikes to citizens promoting and encuraging concrete sustainable mobility policies. Specifically, we developed a web platform that that carried out the whole workflow of e-bike request and assignment in a totally digital fashion, exploiting advanced encryption and Blockchain technologies to guarantee strong authentication and integrity, over the time, of the information entered during the request and during approval. The Use-case described has brought concrete results. Environmental benefits was proven starting from the distance travelled by users.
As expected the use of a public Blockchain increases the overhead in the workflow and in the OTP generation. Anyway, the Public Blockchain represents a strong distributed solution and waiting for a minute could be consider acceptable.

CHAPTER 7

Decentralized Identity Management Osmotic Computing Based

Nowadays Osmotic Computing is emerging as one of the paradigms used to guarantee the Cloud Continuum, and this popularity is strictly related to the capacity to embrace inside it some hot topics like containers, microservices, orchestration and Function as a Service (FaaS). The Osmotic principle is quite simple, it aims to create a federated heterogeneous infrastructure, where an application's components can smoothly move following a concentration rule. In this work, we aim to solve two big constraints of Osmotic Computing related to the incapacity to manage dynamic access rules for accessing the applications inside the Osmotic Infrastructure and the incapacity to keep alive and secure the access to these applications even in presence of network disconnections. For overcoming these limits we designed and implemented a new Osmotic component, that acts as an eventually consistent distributed peer to peer access management system. This new component is used to keep a local Identity and Access Manager (IAM) that permits at any time to access the resource available in an Osmotic node and to update the access rules that allow or deny access to hosted applications. This component has been already integrated inside a Kubernetes based Osmotic Infrastructure and we presented two typical use cases where it can be exploited.

7.1 Introduction

In the Software Architecture field, one of the most disruptive novelty has been introduced by the use of the Micro Service paradigm. Nowadays, This new pattern has been used in

the majority of the Cloud Native applications and this popularity is still increasing thanks to the full compatibility with new other trends related to the containers and the container orchestrations. Consequently to this, some infrastructure patterns have been evolved with the aim of exploiting all the potentiality of Cloud, Containers and of course Micro Services. One of these patterns is called "Osmotic Computing". This pattern is based on ad advanced shape of the Cloud-Edge continuum that allows federating nodes that can host different pieces of an entire application organized in very small microservices and microdata inside Linux Containers, called Micro Elements (MELs).

In a typical Osmotic Infrastructure, MELs smoothly move around the Osmotic Nodes following scheduling rules that can depend on performance, security and availability aspects. These aspects have been already analysed in previous works[84][85], but no one paid attention to the fact that if the MELs migrate, according to the Osmotic Paradigm rules, the accessing rule to the MEL might also need to be up to date. In general, any resource access management is demanded by an Identity Access Management (IAM), which defines roles and roles access to the resources. The IAM is typically a central node with its own SQL database. This kind of architecture is hard to integrate into any Osmotic Infrastructure, which typically is based on a distributed architecture. All the challenges we may meet using a classical IAM in an Osmotic Infrastructure are due to these architectural differences, and they could be:

- impossibility to spread the IAM among different nodes in a peer to peer fashion;
- impossibility to working with some nodes offline;
- impossibility to maintain consistency over the IAMs.

The novelty of our work is represented by the presence of Identity Access Management tailored over an Osmotic Infrastructure that tries to overcome the just mentioned limits, for doing that we will design, implement and test a distributed peer to peer infrastructure, able to work partially offline with a soft consistency, equipped by a set of API that allows to dynamically change the access to any MELs according to the requests of the Osmotic Infrastructure.

The remainder of the chapter is organized as follows. The state of the art is analyzed in Section 7.2. The design of the Osmotic IAM is discussed in Section 7.3. An IAM's prototype is argued in Section 7.4. Two different use cases that might benefit from this IAM are discussed in section 7.5. Conclusion and lights to the future are summarized in Section 7.6.

7.2 Related Work

In the last years, several studies were focused on Security in microservices. The increase of applications based on this architectural paradigm was one of the main reasons to deepen the security in this field. In the research work carried out [86] a recap of last year's studies about security in microservices architecture has been made. In particular, the need to make microservices creation and deployment secure emerges from the work. The research shows that the hotter topics about security in microservices are related to triple-A management. Even [87] solutions about security management in Microservices are treated. The paper has analyzed the security in this kind of architecture at different layers describing how each issue is treated according to the literature produced at that moment. Among the aspects faced, the solution proposed for mutual authentication by Docker Swarm and Netflix is described. The work also talks about the authorization mechanisms used in the microservices architecture. In this field, it proposes some solutions in order to manage the authentication and the authorization of final users in a microservices-based architecture. The solution proposed in the research exploits a token approach by which transmitting the authentication and the authorization among the microservices interested. A similar issue is treated in [88] which the main solutions for end-user authentication and authorization inside microservices use-cases are described. The main solutions, even in this case, foresee the utilization of a token (an example proposed is the JSON Web Token) or, more in general, to consider a dedicated microservice for user authentication and authorization management (an SSO service). The solution proposed, even in this case, is not really innovative and considers the Identity Manager a central element. Other researchers have also discussed the authorization and authentication aspects of microservices. In particular, [89] a solution for authentication and authorization management in microservices is proposed. The architecture described in that work, exploiting the well known defined XACML flow, aims to implement a solution in Machine-to-Machine communication in which the authorization policies are managed. The peculiarity of the solution proposed is the total microservices approach used. Each element, already known in the eXtensible Access Control Markup Language (XACML) standard, is containerized through the already validated Docker technology. The work is interesting because realizes and tests a practice solution in which microservices can manage authentication and authorization not only in a system provided for the end-user. However, it has some limits. It does not face the problem of the Identity Manager in this context. It remains a central element even in the microservices architecture proposed. Security analysis in the microservices context was deepened in

the work [90]. In particular, in that paper, the security was analysed according to different layers. On top of the layers evaluated also the application layer was studied. The solution of authentication and authorization in microservices applications was discussed and some solutions were tested and compared. The solutions considered were a little recap of the most used techniques already discussed. They were related both to the network layer (like IP validation) and to the application one (like secret transmission). The test does not consider some important elements like an advanced authorization management system.
Another interesting work was realized [91]. In this work, a new authorization policy language is developed. It is conceptually based on the XACML standard. It simplifies that standard even using the JSON format to define the rule and the policy. This system allows implementing the main authorization functionalities in a microservices architecture through the "delegation" concept that characterizes the solution presented. Exploiting this new concept, the process of authorization validation can be distributed, among different microservices, without sharing sensible data, within the whole architecture making the solution really scalable and more secure. The case considered does not deep the distribution and the position of Identity Manager inside the architecture. This element should be discovered since the evolution of last year's microservices has faced.
One of the most recent papers, [92], have studied all the research in the security field within microservices-based systems. It reveals and recaps the main topic of the research considered. They are mainly related to authorization and authentication problems like we have already seen.
The work presented in [93], described a new Computational Paradigm strongly based on the Microservices concept, that is Osmotic Computing. That paradigm tries to federate heterogeneous environments (like Cloud and Edge Computing) in order to make transparent a unique "place", to deploy services, called membrane. This federation foresees a possible migration among the nodes federated in the membrane. This paradigm has introduced also an evolution of classical Microservices named MicroElements. MicroElements are characterized by the presence and management of Persistence Data and they are the central element of Osmotic Architecture. Some researchers have focused on the security of the MicroElements inside the Osmotic Infrastructure. The [94, 95] basic principles of Osmotic Computing are defined and the problem of authentication and authorization is presented. In particular, the paper points out the problem related to an authentication distributed in all the federated nodes. The work proposed in our paper wants to solve this problem through the solution proposed.
In this regard, other researchers have focused their work on decentralized management of

identity and access. For example, the research carried out [96] tries to design a decentralized identity system that is based on classical credentials and that exploits Blockchain. It is a conceptually good starting point for Decentralized Identity Management but it is not enough. It is based on classical credentials that are not many safe, and it is based on Blockchain. We can find an evolution of decentralized identity management [97], a distributed Identity Management is considered. In particular, this work takes inspiration from the Self-Sovereign Identity approach [98]. to realize a distributed identity and access manager blockchain-based in which personal data are encrypted through homomorphic encryption. The solution is interesting but it seems a little bit not convenient for the end-user. Moreover, they are not oriented in Microservices and Osmotic approach unlike the solution proposed by us.
A recap of the more important research on decentralized identity is described in [99]. The work shows all the most recent research about decentralized identity management blockchain-based. All the solutions seen within that paper are interesting but all of them are based on blockchains that could represent a third part element inside the Osmotic Infrastructure. The solution proposed in our paper is strictly Osmotic compliant and allows to manage authentication and authorization without third party elements respecting the Osmotic Architecture itself.

7.3 Design

Typically IAMs, are deployed using an architecture similar to the one shown in Figure 7.1. In the figure, we represented two different nodes. The first node acts as a centralized IAM, composed of a set of access rules and user and service data stored in a database, a set of APIs for authenticating users and a proxy server used to forward the requests from the client to the correct service that instead are stored in a second node, usually located in a private network area. Typically access rules are represented like a sentence with the following shape: "⟨ Subject⟩ can ⟨action⟩ on ⟨object⟩", in these cases the actions is typically involved in CRUD (Create Read Update Delete) set operations and the object is any available resource. The subject depends on the security pattern followed, for example, RBAC or ABAC. This topic is out of the scope of this work and for simplicity, we will consider starting now an RBAC based IAM.

In this scenario, users use the IAM's API to authenticate themself using their credentials, and the IAM's proxy to reach the services. The proxy will use the authentication information

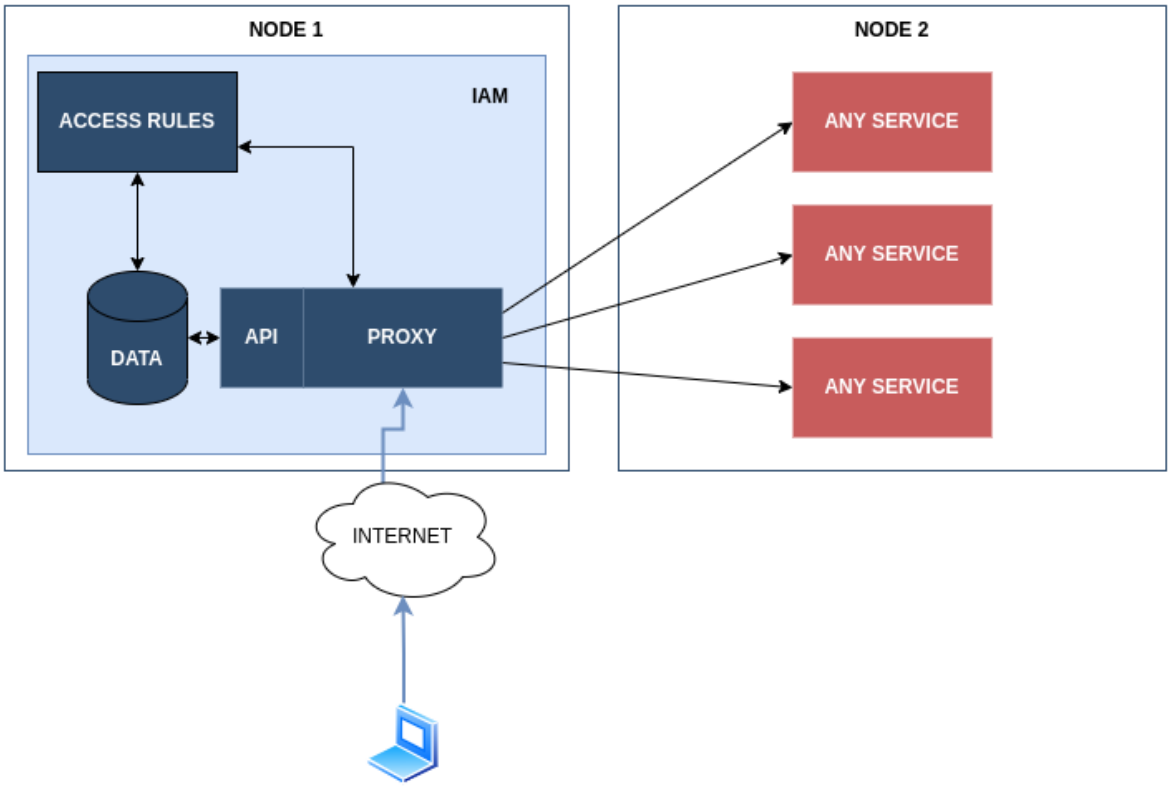

Figure 7.1: Centralized IAM Architecture

provided by the client and the access rules stored to decide if they can reach the requested resource or not. This flow is well described in the sequence diagram in Figure 7.2. As we see there are no interactions between the clients and the real requested resources, all the messages in fact arrive at the Proxy and only the authorized ones are forwarded to the resource. In an Osmotic Infrastructure MELs are spread among different Osmotic Nodes as well as other Osmotic Components. Often, Nodes are not reachable between them, due to network disconnection or security reasons related to the Osmotic Membrane [93]. Due to that, tailoring a centralized IAM in this scenario is a high risk, since it may affect the usability of some applications inside the Osmotic Infrastructure. For these reasons, we designed a full distributed IAM as shown in Figure 7.3.

In this new scenario, we have many Osmotic Nodes, and in all of them, an IAM is installed. The IAM is still composed of the elements seen before, a database containing the user's and resource data, a set of access rules, a set of APIs and a proxy service.

This time, the IAM is equally distributed among all the Osmotic Nodes, in a peer to peer model. Each node must be able to work even if any of the others are unreachable, for this reason, each IAM is in charge to guarantee secure access only to the local services installed in the same node where the IAM is, and eventually to the others only if they are reachable, in

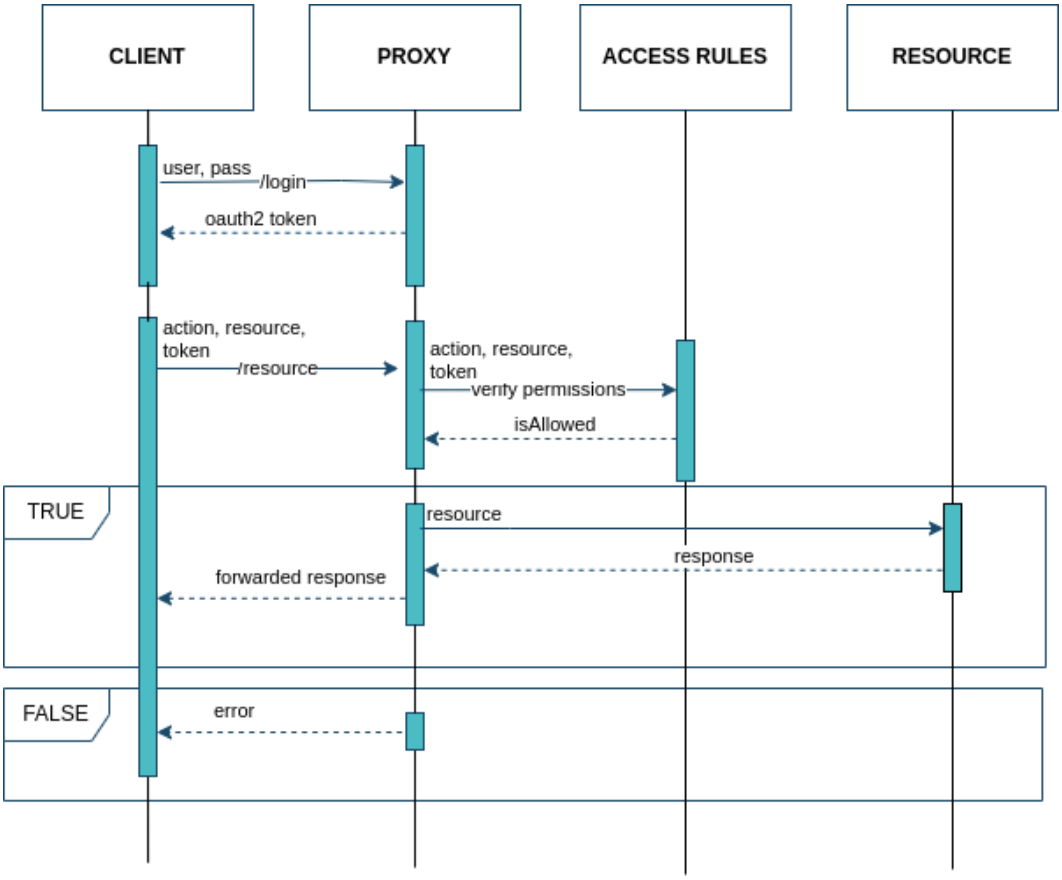

Figure 7.2: Authorization and Access Flow

this way a service becomes unreachable only if the entire node is unreachable. The different IAMs instances need anyway to be synchronized because they need to know the rules to access to service they are not hosting at this moment but they could later. To guarantee this consistency, access rules, user and resource data are stored in a peer-to-peer database that accompanies the proxy. The consistency between the database can be guaranteed using must use a multi-version concurrency control pattern (MVCC). Finally, the data updates happen using the APIs the IAM is provided. These APIs can be consumed by the MELs or by the Orchestration Engine, this depends on the implementation of the Osmotic Infrastructure, nevertheless the implementation, each API instance is authoritative only for the Node where it is installed, this behaviour prevents any conflict during the data synchronization without compromising the usability of neither infrastructure and applications.

This approach requires that we know the IP addresses of the Osmotic Nodes where the services we need to contact are hosted, for usability constrain we cannot accept this. For this reason, we put in our architecture a further layer represented by a Load Balancer. This service can be deployed in replica in an SDN in order to guarantee high availability. The Load Balancer just redirects the requests coming from a client to the correct service's authoritative

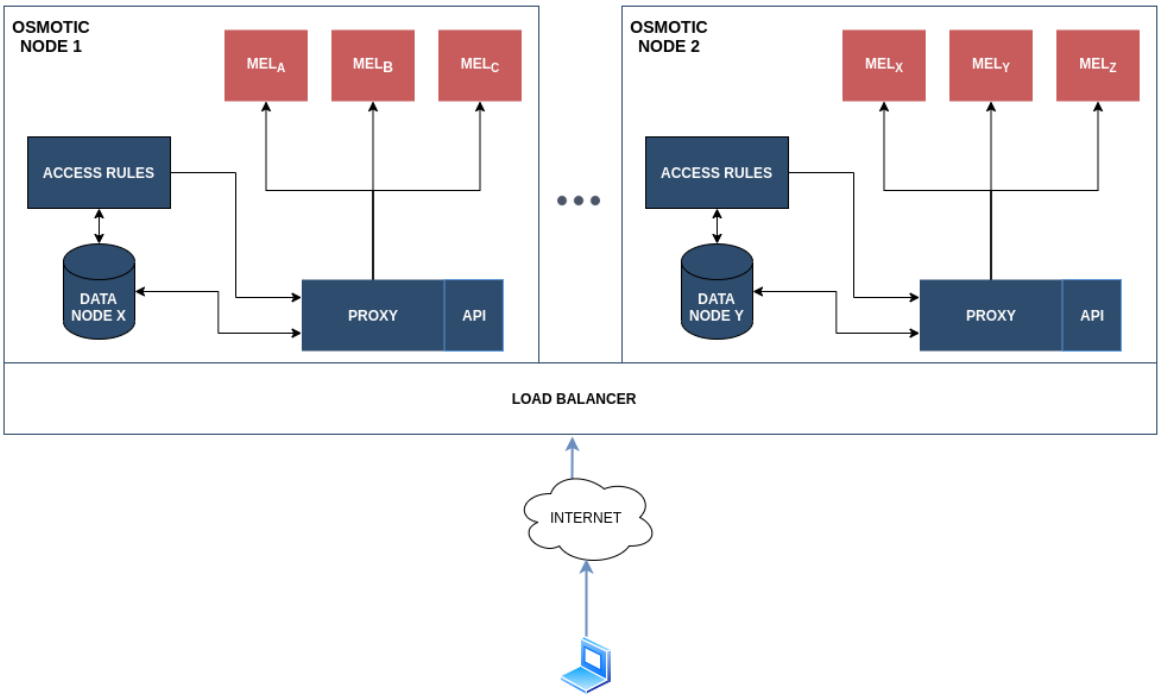

Figure 7.3: Distributed IAM Architecture

node, guaranteeing transparency to the final user. Finally, thank to the use of this element we can easily reach any available Osmotic Node without caring about their real IPs. From a client-side point of view, this new full distributed approach is transparent, in fact, the same flow described in Figure 7.2 is still valid. In the distributed approach, in fact, the client still contacts a single access point which is the Load Balancer, this element will forward the request to the correct resource authorities proxy, which will manage the authentication and resource's access flow.

7.4 Implementation

Following the designed architecture shown in Figure 7.3, now we are going to implement it, in order to obtain a full working prototype.

About the Osmotic Infrastructure, we trust in an Edge Kubernetes infrastructure. Thanks to this engine we can easily deploy MELs using Pods, and we can orchestrate Pods using the Kubernetes APIs. We managed the Osmotic IAM as a multi-container pod instanced like a daemon, namely a Pod deployed in every node of the cluster.

We implemented the IAM using a distributed peer-to-peer database and a full stateless process able to interact at any time with the local database instance. This process is in charge of exposing API to interact with the security roles stored in the database and to proxy the

requests towards the requests MELs (once their access is permitted).

7.4.1 Database

Finding a database able to meet our requirements is not easy, since most common and used SQL and NoSQL databases horizontally scale only using a master-slave model, like MySQL and MongoDB for example, and often this means that all the writing operations pass through the master and all the reading operations might need to be approved by the Master. This approach does not fit our model since we need for writing and read to/from any node at any time. A good choice for our requirements is represented by CouchDB by Apache. CouchDB is a document-oriented database, that can be deployed inside a Cluster using a peer-to-peer approach. Each peer can contain a part of or an entire dataset, but it does not require keeping a stable connection with the other peers continuously, data in fact are considered eventually consistent, because peers will synchronize them as soon as possible, without disallowing updating during network disconnection periods. Once the peers are reachable again, they will use a consensus protocol that exploits the MVCC data model to keep the last updated documents.

Finally, we are going to use CouchDB to store the accessing rules the resource and the user information.

7.4.2 API Set and Proxy

API and Proxy are both web services that need to interact with the same data components and with the same transient data, for this reason in the implementation phase we merged the API server and the Proxy into a single service that acts all the IAM operations we described until now. This service needs first of all to authenticate and authorize users, and therefore to update users' data information inside the CouchDB instance hosted in the same node. The service authenticates using a Role-Based Access Control and Cookies to maintain a connection state. The APIs involved in this are identified as Authorization API.

Authorization APIs are used to guarantee secure access to the Security API. This new set of APIs is in charge of updating the Access Rules which the node is authoritative.

Finally, Authorization and Security API are used by the Proxy for processing a request coming from the internet. First of all, the Proxy fetches the cookie provided in the request and sends it to the Authorization API which will reply by providing the role of the user who owns the cookie, this role will be forwarded to the Security API with the address of the requested

resource, next the reply will tell the Proxy if the request is authorized or not. Finally, the proxy will forward the request to the resource or will reject it, sending it to the client in the first case the service's response or an error response in the second case. We implemented this component exploiting NodeJS, Express, Nano and SuperLogin packages node.

1. ExpressJS is a very famous framework used to manage an HTTP server using NodeJS; it is used to authenticate and authorize users, expose API for updating the accessing rules and finally for proxying the requests to the MELs.

2. Nano is the official CouchDB's Javascript library that allows database interaction;

3. SuperLogin is a quite popular microframework for Identity Management based on CouchDB and NodeJS; it works also like Express middleware, enabling the authentication and authorization policies (customized by us in order to interact with the dynamic rules).

7.4.3 Deployment

The distributed IAM needs of course to be integrated with the Osmotic Infrastructure realised using Kubernetes. The proxy and the CouchDB node instance are strictly dependent and they cannot work if the other one is stopped. For this reason, we used a multi-container Pod to deploy the IAM, composed of the CouchDB instance, the Proxy instance and a single run configurator that ensured that the proxy and database are well configured.

As we mentioned before, we need to deploy a single IAM instance in any Osmotic Node, for doing that Kubernetes provides a sort of Deployment Method called DaemonSet. "Like other workload objects, a DaemonSet manages replicated Pod groups. However, DaemonSets attempts to join a one-pod-per-node model across the entire cluster or a subset of nodes. When you add nodes to a node pool, DaemonSets automatically add Pods to the new nodes as needed"[100].

7.4.4 Load Balancer

The Load Balancer is strictly related to the Osmotic Infrastructure implementation, in this case, Kubernetes provides the Kube-Proxy which is installed on each Kubernetes cluster's node and acts as a proxy for UDP, TCP and SCTP communications. It is used as the entry point for reaching the Osmotic Services, for this reason, any other further design choice like

the use of a service's authoritative IAM is demanded from the IAM itself instead of from the Load Balancer.

The final result is shown in Figure 7.4.

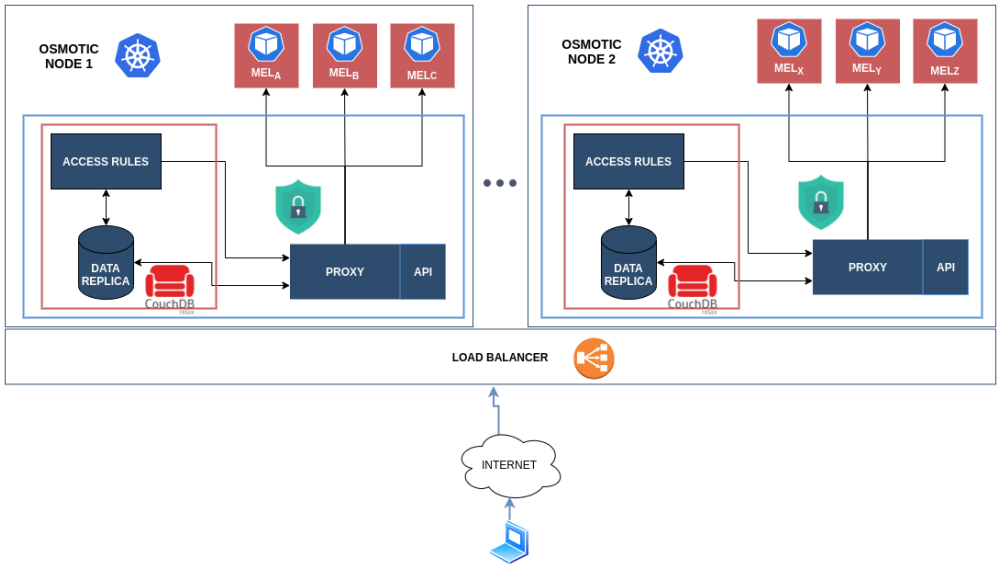

Figure 7.4: Osmotic IAM

7.5 Use Cases

In this chapter, we are proposing a solution for guaranteeing secure access to any application deployed inside an Osmotic Infrastructure. To keep secure access to these applications, we designed a Proxy that through a series of rules shared among all the nodes and stored in a local database that in an RBAC fashion allows or denies access to the resources.

The first concern about Osmotic Computing regards the policy dynamic, in fact, in an Osmotic Infrastructure any access rule might fast change due to service or network constraints, for this reason, we provided our Proxy with a set of APIs that allows adapting the proxy to the new rules.

The second constraint in Osmotic Computing is network disconnection. In a fully distributed environment, sometimes some nodes could be isolated from the other ones, but this should not limit the use of the node itself. Since nodes share policies and data they should also be able to synchronize them as soon as they can interact again each other. For overcoming this limit, our IAM has been also designed in an Osmotic fashion, decoupling it into as many replicas as the nodes are and using a peer-to-peer database able to maintain an eventual consistency and to synchronize it with the other database peers as soon as possible, without

denying the usability.

In this section, we will present two simple use cases where Osmotic Computing is applied where the just-mentioned problems might present and how our solution solves them.

7.5.1 Smart City Use Case

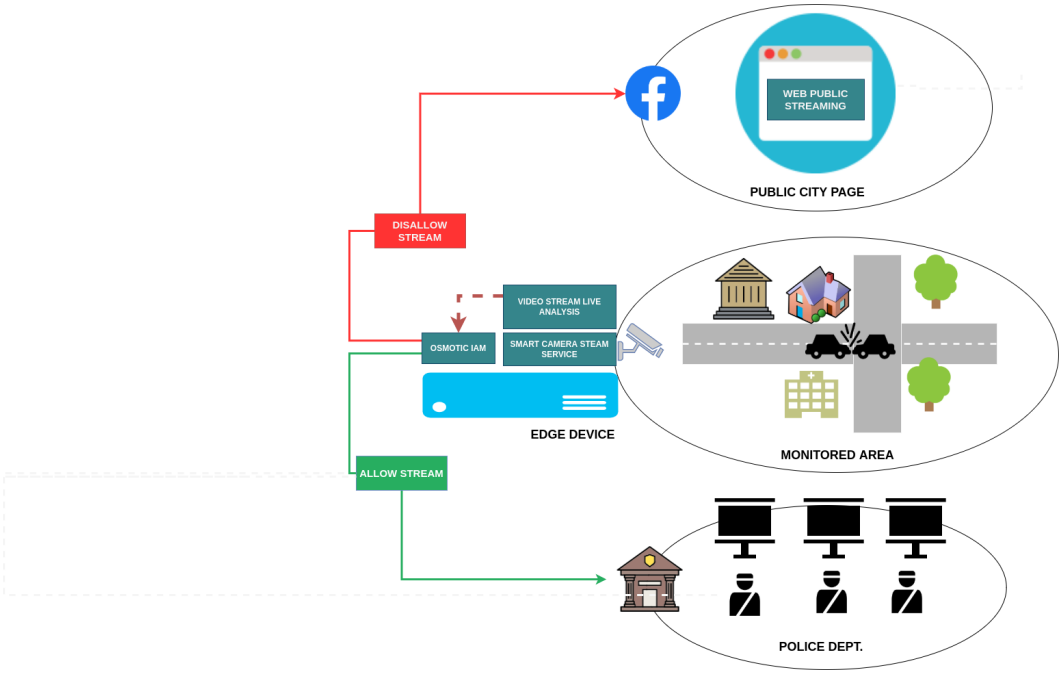

Figure 7.5: Smart City Use case

Nowadays, Smart Cities are characterized by many common features, like distributed access points, environment monitoring, tourism and security tech-services. Smart video stream services are for example, used to show some important city areas, museums, or parks directly from the smartphone or video panel spread around the city. Cameras and video streams are also used to maintain some areas under control, and usually, these cameras are used by the Municipality's policy or any other security department [101]. Of course, in the first case, access to the camera can be guaranteed to any public with or without authorization, instead, in the latter case, only the public with proper authorization should access the video stream. These policies are quite static, but what happens if, under the eye of a public camera a crime happens, showing on a grand scale sensitive contents? Unfortunately, this could be the case of some terrorist attacks that happened in the last year in Europe, where the images traveled around the World and have been shared on some social like Youtube and Facebook. As proof of concept, consider Figure 7.5; we are describing an Osmotic node installed near a city area with a cross street. This osmotic node is running a smart camera that records the

scenes and a video stream analyzer that recognizes the recorded events. The streaming video is shown on the city's public Facebook page. Unfortunately, an incident happens; hence the video stream analyzer recognizes this event and immediately updates the Osmotic Proxy installed in the same Edge node, restricting the video stream access only to Public Security City's officers and disabling the video stream on the city webpage. The use of an Osmotic IAM could have solved this kind of inconvenience; in fact, as soon as a suspicious event would have been detected in the video stream, automatically, the access to the video stream resource would have been updated, and the camera would have been used only by a strict number of people involved in the city's security avoiding any dispersion of sensitive content.

7.5.2 Rural Area Use Case

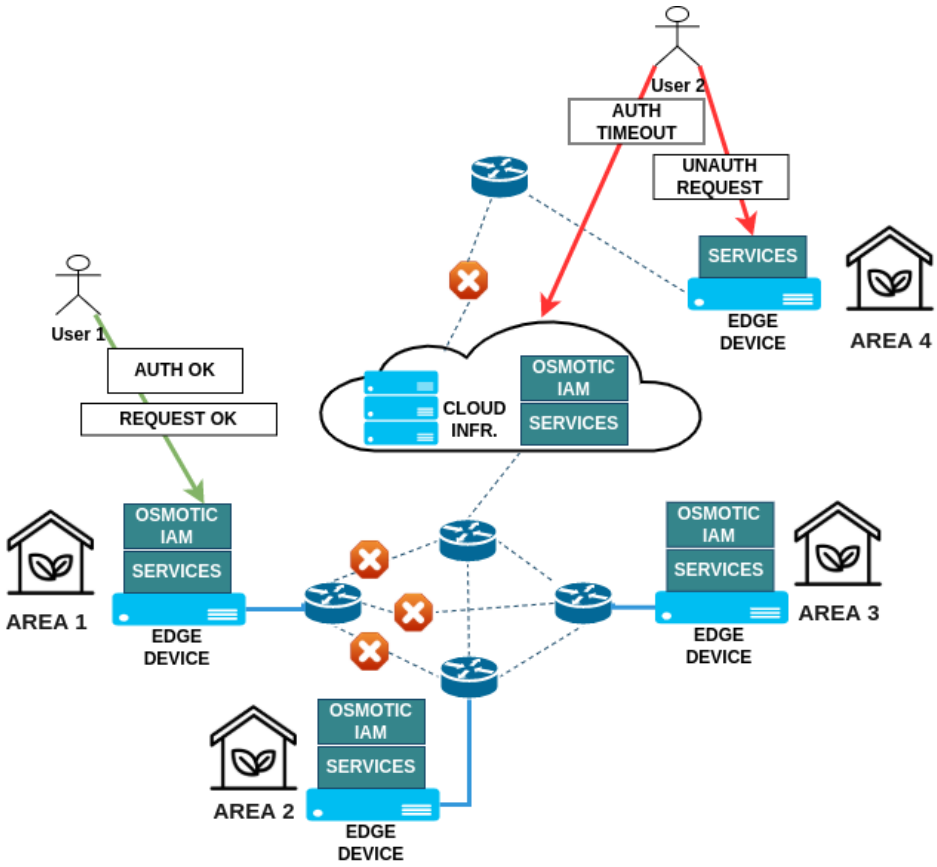

Figure 7.6: Rural Area Use Case

A rural area or countryside is a geographic area that is located outside towns and cities, where typically the provided services are very poor, and the connectivity is quite limited. An interesting research topic that is born in the last years regards the possibility of enriching these areas with cheap Edge-based services able to provide first-need raw services for healthcare, industrial automation, and document sharing. In this kind of space, typically are deployed

meta-industrial services, or first-need services, like document sharing with other areas in the same environmental conditions.
The work we developed in this chapter finds a concrete case in this situation. As we already said in our work, Identity, and Access Management is traditionally centralized. The approach proposed in our work goes beyond this concept. By decentralizing these operations, our solution can represent a valid alternative to the reality we are considering. A hypothetical private resource can be protected in a Rural Area even if the Network service is not always efficient. In the typical situation in which Authentication and Authorization are centralized, the Network service absence could affect the resources' private access. Indeed, the typical systems, in this situation, do not allow resource access because they cannot verify the correct roles of the users and, in general, the authorization operations cannot be performed. In a real use-case in which our solution is applied, the Authorization could not be conditioned by the global reachability of the local nodes. For example, we can imagine different systems IoT-based like a typical Smart Agriculture context that descends private data to the Edge Layer. The data detected could not be available for all entities that can physically access the Edge because, for example, they could be related to a particular Business advantage. Figure 7.6 shows the concrete context. In Area 1, for example, the user wants to access a particular resource reachable in his area only through a local connection because Internet access is not available. In typical situations, the Identity and Access Management are not able to determine the authorization attributes or the roles of the user, and it is not able to understand if the user can or not access the resource because, normally, the Identity and Access Manager are centralized and placed in a remote server only reachable via the Internet. As shown in the Figure, the architecture proposed in this work solves this issue.
The Osmotic IAM solution we have described in this chapter could be applied on the Edge Layer, relegating to the latter Authentication and Authorization operations. Even if the Local Edge devices are not reachable worldwide, they can anyway authenticate and authorize the permitted users.

7.6 Remarks

Osmotic Computing is a very interesting research topic that aims to integrate Cloud, Edge, Fog and IoT in a unique Computation environment where applications are free to move following different concentration rules based on many internal and external parameters. This goal is really ambitious because can enable many different use cases related to Security,

DevOps and Smart Cities, but due to its complexity, many limits are still not solved at all. Two of these limits regards the access policies for the application and users and the resilience to network disconnection, the Osmotic Computing paradigm, in fact, foresees that an application can evolve, changing the node that hosts it, the active configurations (Micro Data) and the consumers. From an architectural point of view Osmotic aims to distribute the applications among different nodes with the equal role, but different characteristics. This fully peer-to-peer design could fall into partial network disconnection, where only a few nodes can communicate with each other and in many cases, this represents a reason for some trouble. To overcome all these limits we presented a new Osmotic Computing Component which acts like a Distributed Identity and Access Management. The IAM is itself a peer to peer component thought to be run in every node of an Osmotic Infrastructure. It is designed to be resistant to any network disconnection, in fact, each IAM instance works independently from the other ones, but synchronizes with others' access rules and data as soon as the network allows, therefore it keeps working at any time and guarantees authorized access to the Osmotic Application at any time. To guarantee dynamic access policies for accessing the Osmotic Applications, we enriched the IAM with a set of APIs authoritative for the node where they are hosted, these APIs allows updating any rule inside the IAM and they can be consumed by the Osmotic Engine, by the application or by external but trusted clients. Immediately, once a rule is updated the IAM reflects it in the Proxy used to access the resources, and in parallel, these rules are sent to the others node to keep synchronized over the entire infrastructure. This chapter is an entry point for several future works we are already planning in the field of Osmotic Computing and Security. First, in this work we have taken for granted the use of any Osmotic Engine, citing some work proposed in this field, but we aim to work on an entire and well-tested architecture where the Osmotic IAM directly interacts with the Osmotic Engine we are currently developing. Furthermore, this work opened the possibility to think about a way to decentralize sensitive data by the use of local edge databases peer and cloud peer databases, thus we are planning to use some concepts of our design to keep private data safe on edge and synchronize them only when needed.

CHAPTER 8

Federated Learning Solutions to Decentralize Computation on Safe Data

Federated Learning represents among the most important techniques used in recent years. It enables the training of Machine Learning-related models without sharing sensitive data. Federated Learning mainly exploits the Edge Computing paradigm for training data acquired from the surrounding environment. The solution proposed in this chapter seeks to optimize all the processes involved within a Federated Learning client through transparent scaling across different devices. The proposed architecture and implementation abstracts the Federated Learning client architecture to create a transparent cluster that can optimize the complicated computation and aggregate the data to solve the heterogeneous distribution issue of the data in Federated Learning applications.

8.1 Introduction

Although Cloud Computing has been a great innovation over the years, changing the way computational resources are provided, over time some of its limitations have been molten. Cloud computing in particular has shown weaknesses concerning data security and data privacy. This issue becomes even more important when we think about use cases such as healthcare, which are very common in this paradigm. For this set of reasons, over the past few years, research trends have been pushing the paradigm of Edge Computing, which contrasts precisely with the centralization of the Cloud by trying to overcome its limitations. Edge computing, in fact, moves computation to the *borders*, near where data is generated

or collected, i.e. IoT devices. Considered the main purpose of Edge Computing, it makes use of devices with limited resources. Over time, however, advances in technologies are improving these devices, making them increasingly high-performance. This is therefore driving research and new architectures to move computation right into Edge Computing so that all its advantages can be exploited. The main advantage is the security ensured by Edge Computing and by its nature.

For this reason, Google introduced in 2017 an innovative technique that allowed the training of Machine Learning models without compromising its users' data privacy named *Federated Learning* (FL).

In the classical approach, to train a Machine Learning model, the data was shared with a Cloud Infrastructure increasing the risk that the latter could be violated. In this new approach, each Edge Device trains a local model and all the models will be aggregated without sharing data with Cloud. Various advances have characterized federated learning, which, however, still has some open problems. Some of them are caused by the several computational constraints still present in Edge Device.

For this reason, in this work, we propose a new approach to the Federated Learning architecture design in which each client is not necessarily a single Device. Our solution tries to decouple the data analysis flow present in each Federated Learning client, distributing the main tasks across more physical devices. Our solution can overcome the problem of computational constraint in Federated Learning because, the single client is, in our solution a cluster of different physical devices. In this research we propose a new architecture, implement it and we tested it in order to prove the advantages this new approach can bring. The major contributions of this chapter are the following:

- we consider the current solution in Federated Learning, especially about Clustering approach;
- we discuss the challenge of Federated Learning current architectures;
- we propose our new approach in Federated Learning client design
- we implement the solution proposed testing it and comparing it with the current, single-node, approach.

We made our experiments for different dataset sizes (from 500 samples up to 10000), considering a FL client composed of two different devices (the *training node* and the *data node*). Results showed that the benefits of our approach increase at the increasing of the dataset

size. The rest of the paper is structured as follows: a review of the literature is presented in Section 8.2; a description of the proposed solution is presented in Section 8.3 followed by its implementation Section 8.4; an assessment of the proposed system is presented in Section 8.5. Finally, conlusions an light on the future are presented in Section 8.6.

8.2 Related Work

8.2.1 Background

Federated Learning (FL) is one of the most innovative techniques developed in the last few years. It was designed and created by Google in 2017 in order to perform Machine Learning activities over data without sharing them with a Central Server Infrastructure [102]. Indeed, this innovative approach distributes the training of a Machine Learning model among the Edge Clients got by users themselves. Each user's device trains a partial model that an aggregator entity will aggregate. The first aggregation algorithm introduced was *FedAvg* [16]. The latter exploits the Distributed Selective Stochastic Gradient Descent [103]. This approach aggregates the partial models trained by all devices performing an average of the weights trained to get a global model that is re-transmitted to each client.
According to the first definition of FL, the aggregator entity is a Central Server, often placed in a Cloud Infrastructure. Several research works have decentralized the aggregation of partial models to exploit the advantages of a decentralized architecture [104][105]. This new approach eliminates the central aggregator node envisioned in the first definition of FL in order to avoid a single point of failure, optimize network communications, etc.

8.2.2 Decentralized FL

The decentralization strategy moves the aggregation process from the central server to the edge nodes themselves.
Different architectures and different strategies have been tried to optimize the traditional federated learning architecture distributing the aggregation process according to different patterns [106].

- Hierarchical Architecture. It introduces different layers to decouple the aggregation of the nodes involved by maintaining a central node for the final aggregation [107] [108] [109].

- Regional architecture. It is similar to Hierarchical Architecture but the aggregation is no longer placed in a central node, but it is performed within the edge layer assigned to the respective regions [110] [111].
- Full Decentralized Architecture. This pattern involves the aggregation process in the final edge nodes themselves. Each top layer where aggregation used to occur is now removed and aggregation is implemented in the nodes themselves, which according to different strategies exchange the trained partial model, i.e., the weights [112].

Decentralized architectures, as seen, also delegate to end nodes operations that were previously in charge of the centralized aggregator node, like the aggregation itself. This increases the computational load of the edge node located at the extremes of the architecture.

8.2.3 Clustering

One of the most used approaches in the literature (strictly related to the solution proposed in this chapter) was the Clustering of FL clients. This kind of approach tries to aggregate a specific number of clients in order to overcome some well-known problems like the bandwidth usage in the communication of the models and the Non-independently and identically distributed data (Non-IID) [113] [114]. The cluster can be designed according to several criteria. These could be, for example, the location of nodes, their energy impact or the distribution of data [115]. Other works create dynamic clusters in order to improve the accuracy of the model and optimize the aggregation operation [116]. However, all research work on this subject deviates from our solution, which is at a lower level. In contrast to the works cited above, our proposed solution does not consider each physical node necessarily a trainer. As we will see in detail, the cluster we envisioned wants to abstract the federated node itself so as to optimize training, aggregation, and side operations. The starting point of this work is a research that tries to abstracts the Edge Devices according a Edge as a Service paradigm that is considered propaedeutic to Decentralized Federated Learning [117, 118].

8.3 Solution Proposed

The architecture proposed consists of a cluster of N nodes representing a single FL client. The solution, therefore, aims to propose a new approach to the design of the main component present in a Federated Learning architecture.

8.3.1 Architecture

The proposed new approach tries to distribute the computational load over several physical devices through the use of precise architecture and precise workflow. Clustering of each FL client allows for exploitation of more computation and allows better management of collected data and trained local model.
In the cluster, two types of nodes are considered:

- the training nodes that actually train the local model.
- the data nodes for collecting and managing data involved in the training model.

In order to distribute each task inside the devices belonging to the same cluster we packed each function inside a container [119]. In this way, the number of devices for each type is not defined *a priori* but could be managed with respect to the specific use case. The cluster provides more computational resources for each operation.
The training nodes can better exploit the hardware for training operations while the data nodes can only take care of data collection and their pre-processing.
The replica of data nodes, within the cluster, allows for uniform data collected avoiding that

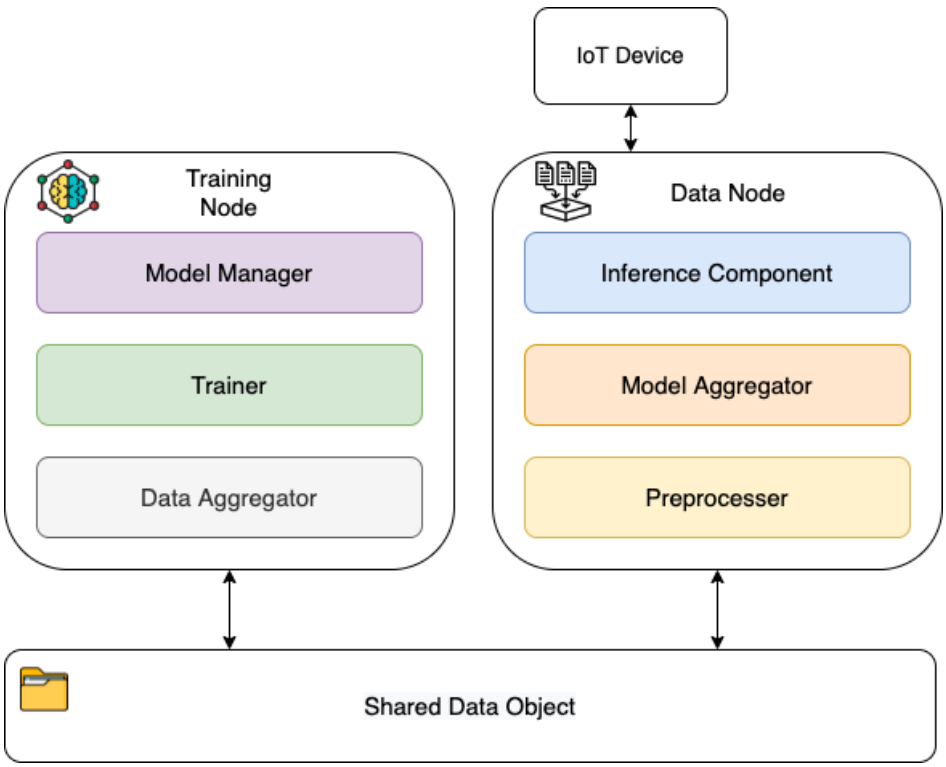

Figure 8.1: FL Architecture with clustered clients

training with Non-independently and identically distributed (Non-IID) data could degrade

global model accuracy. The Figure 8.1 depicts how the nodes are organized.
Three main components characterize the training node:

- The Trainer is the component that carries out the local model training.
- The Model Manager is the component that manages the local model trained (e.g transmit the locally trained model parameters to the other FL components for the aggregation).
- The Data Aggregator is the component that triggers the training phase when the dataset is prepared. Moreover, it aggregates and uniforms all the data coming from Data Node.

The Data Node, as we said, is in charge to collect data and preprocess them in order to prepare the dataset through which the training phase will be accomplished. As Figure shows, it can be enriched by an IoT Device and it is constituted by three main components:

- The Model Aggregator that carries out the local models' aggregation of other clients.
- The Inference Component that makes the inference with the global aggregated model.
- The Preprocessor manages the collected data in order to prepare them for the training phase.

The Figure 8.1 shows another important component: the Shared Data Object. This represents the entity by which, within the cluster, the Nodes share data with each other. Hence, it is in charge to store the dataset preprocessed by Data Nodes and exploited during the training phase.

8.3.2 Flow

In our solution, as we can see from the proposed architecture we imagine a precise workflow that includes the components described. This workflow generalizes the main steps we have in federated learning. As we have seen, the literature has introduced many algorithms for optimizing the training of the local weights of each device and in their aggregation across the entire federated architecture. Our proposed solution is transparent with respect to these solutions. It investigates at a low level the management of operations leading to the training of the partial model. For this reason, we have defined a specific flow that generalizes them and which, in our solution, we distribute optimally. We can summarize the flow in the following steps:

1. The data collection performed by IoT Layer

2. The preprocessing of collected data for the training phase
3. The aggregation of all data collected
4. The training of a partial model that will be shared among the whole architecture
5. The aggregation of the partial model received by other FL clients

Subsequent to the steps described, it is necessary to include any eventual inference that would occur through the aggregate global model.
The proposed solution decouple these steps so that they can be optimized, parallelized, and distributed over multiple computational resources. The components described are in charge to perform this steps in order to carry out a total local training. To do this, the architecture components described act by implementing the flow through synchronous communications that enable the creation of the local model. The Figure 8.2 shows just how the flow unfolds

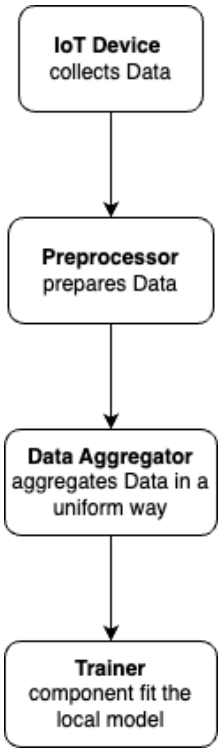

Figure 8.2: Steps involved in a local training

and how each step is performed by the components described above. The proposed solution therefore not only builds an architecture, but establishes a protocol that orchestrates all the steps within the cluster.
As previously discussed, the work carried out in this chapter allows to overcome the unbalanced Non-IID dataset in Federated Learning. Indeed the Figure 8.2 shows us, among the

other steps, the aggregation of Collected Data. In this kind of aggregation, all Data collected by different Data Nodes are aggregated in order to prepare a dataset that can be uniform with respect to the other partial dataset of Federated Learning architecture. This kind of approach is analogue to a solution seen in literature [114]. Unlike that solution, the partial dataset is not provided by a Cloud Infrastructure but it's provided by different Data Nodes of the cluster. The Data Aggregator components can establish how the dataset will be trained in order to uniform the size ensuring homogeneity across the whole FL architecture.

8.4 Implementation

In this section, we will see how the described architecture was implemented and what technologies were used. In our work, we carried out a prototype implementation of the above architecture. Specifically, we realized a cluster of edge nodes capable of training a partial model by completing the entire flow starting from data collection and ending with training the model itself.

8.4.1 Infrastructure

The infrastructure in which the implementation was implemented consists of microprocessors, which as is well known, represents the main devices used in Edge Computing. Specifically, we considered in our implementation, a Raspberry Pi 4 with 4 GB of Ram for the implementation of a Data Node and a Jetson Nano (Developer Kit) with 4 GB of RAM and integrated GPU for the Trainer Node.
Both devices were equipped with a Linux distribution, Raspberry OS and Ubuntu 16.04, respectively. The devices equip an arm 64-bit architecture. The cluster implementation was done through the configuration of a Kubernetes cluster. Kubernetes represents one of the major solutions in container orchestration within a device cluster. In particular, k3s Kubernetes distro has been used [100]. k3s was a lightweight Kubernetes distribution developed by the Rancher organization. Due to its lightness, k3s is strongly recommended for Edge Devices. However, it looks like very flexible because it supports all kinds of architecture. In our case, we exploit k3s in order to create an Edge Devices Kubernetes Cluster. Inside our cluster, we have deployed all the components above described. As we have shown in Section 8.3, another important component in the architecture is the Shared Data Object. The latter is the main element by which the cluster can be considered like one single Federated Learning client. Indeed, the Shared Data Object allows the physical nodes to share all the contextual

data, like the collected IoT Device Data, the preprocessed data and so on. GlusterFS tool was used to realize that. GlusterFS is a network filesystem that allows the creation of distributed and replicated filesystems among different devices connected with each other [120].

8.4.2 Architecture Components

Architecture Components, that has been described in Section 8.3, were each implemented as a microservice within our cluster. In particular, each component was built as Container Image and it was deployed as a Kubernetes Deployment inside our cluster. The components are almost always implemented as HTTP servers because, as we have said, the cluster perform a synchronous flow to train a local model. In particular we have

- The Trainer was implemented using TensorFlow framework and Keras API. The component takes the data from Shared Data Object after they are written from Data Aggregator. It load and trains an existing model saved on Shared Data Object.
- The Model Manager is a Python script that sends the new local model trained to another entity. It can be a cloud aggregator server or, in a decentralized fashion, another Federated Client. It is triggered by Trainer component
- The Data Aggregator is implemented as a infinite loop that check the new data written by Preprocessor component.
- The Model Aggregator is realized as a script that implements the Aggregation Algorithm. This step can be implemented, according to the literature, in several ways but the actual implementation is out of the scope of the work proposed here.
- The Preprocessor Component is the component that preprocesses the data acquired via IoT devices. Even in this case the practical implementation depends by each use-case. We won't deep that.

The inference component was implemented too. It is not present within the whole Flow. It is implemented, even in this case as a HTTP Server. It's not called by internal element but it is external exposed.

Listing 8.1: Training Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: training-app
  namespace: arch
spec:
  selector:
    matchLabels:
      app: training-app
  template:
    metadata:
      labels:
        app: training-app
    spec:
      containers:
      - name: training-script
        image: alecatalfamo/training:0.1
        imagePullPolicy: Always
        volumeMounts:
        - mountPath: "/home/app/gluster"
          name: task-pv-storage
      nodeSelector:
        device: jetson
      volumes:
      - name: task-pv-storage
        persistentVolumeClaim:
          claimName: gluster-claim-2
```

The Listing 8.1 shows one of the YAML files used to deploy all the architecture inside the Kubernetes cluster. The code shows us, in particular, the deployment of the Training Component inside our Trainer node. As we said, the train component, in our prototype is realized through a Jetson Nano. For this reason we can see that we have specified a particular node for deployment labeled in our cluster with "device=jetson". The other elements of the Listing are the main paramenters foreseen in the Kubernetes Deployment API. Obviously, all the components implemented were developed using Kubernetes and a YAML file. We report exclusively the case of training for simplicity. The structure is the same for other components.

```
1 def watchdog():
```

```
    while True:
        if checkAllClassesFiles(N):
            for cla in classes:
                pathWrittien =
                arrayWrittenPaths[cla]

                pathTrain =
                arrayTrainingPaths[cla]

                copyFilesToTrainingFolder
                (pathWrittien, pathTrain)
            try:
                startFit()
            except Exception as ex:
                print(ex)
                print("Fit Problem")
                continue
        else:
            continue
```

Listing 8.2: Data Aggregator triggering function

The Listing 8.2 shows a part of Data Aggregator. In particular, it shows the main functionality of this component. The listing shows part of the implementation of the Data Aggregator component, and in particular the function of *watchdog*. It takes care of calling the Trainer only when in the Shared Data Object, which in our case is represented by a folder replicated via GlusterFS, there is a specified and pre-configured number of files.

8.5 Performance Evaluation

The main innovation of our work is the distribution of the main tasks involved in a Federated Learning Client. The main advantage of our solution lies in the distribution of the tasks involved in Federated Learning. Our performance tests will focus on the overhead that can be introduced by the necessary networking. We will also go on to measure the benefits, mainly in terms of timing, that distribution can bring to model training. This in fact represents one of the most onerous tasks for an Edge device that, as mentioned, is usually limited in computational resources. To test the performance of our solution, we considered our prototype implementation, which as we recall consists of a Raspberry PI 4 of 4GB and a Jetson Nano (Developer Kit) equipped with a GPU and with 4GB of RAM. In order to

test we have implemented a Neural Network Model for binary recognition. In particular,

Figure 8.3: FL Architecture with clustered clients

as we can see in Figure 8.3, we have tested the epoch training time comparing the situation in which all the tasks are distributed in two nodes and the situation in which we have one single node that manages all the tasks. We have tested the training, increasing the number of samples used for the training. In particular, we have compared the situation in which the deployment is carried out in a single node and the situation in which these are distributed across two nodes: The Data Node and the Trainer Node. As we can see the result obtained is an increase of the training time for the single node context. Obviously, this comparison can appear not meaningful, but we have to say that the trend can improve if we increase the node in the cluster. We have tested a cluster of two nodes. We can imagine implementing, as said, even clusters with a greater number of nodes. The Figure 8.4 shows what is the average memory per node, used during the training of the local model. In particular, we can see that the average of used memory per node, is lower in our solution in which all the flow is managed by a cluster. In our node indeed we have more processes that have to manage all the flow and the cluster but the average used memory in a node is still less respect to the solution in which we have a single node. The Figure 8.5 expresses a similar concept. If we consider, as a percentage, the memory used compared to the total memory available, we can see that the tests confirmed what was already guessed. Our solution provides a greater quantity of computational resources for the management of the whole flow.

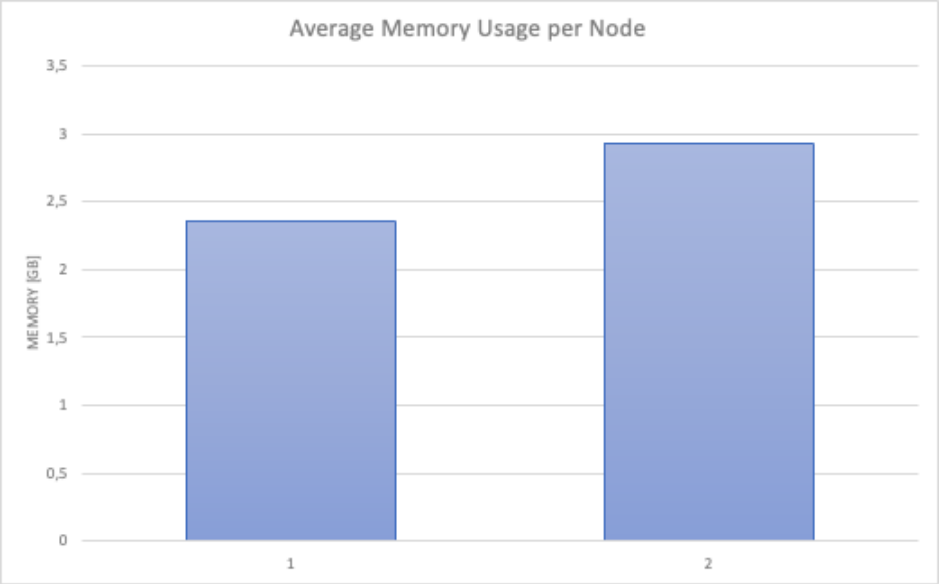

Figure 8.4: Average Used Memory per node in Training phase

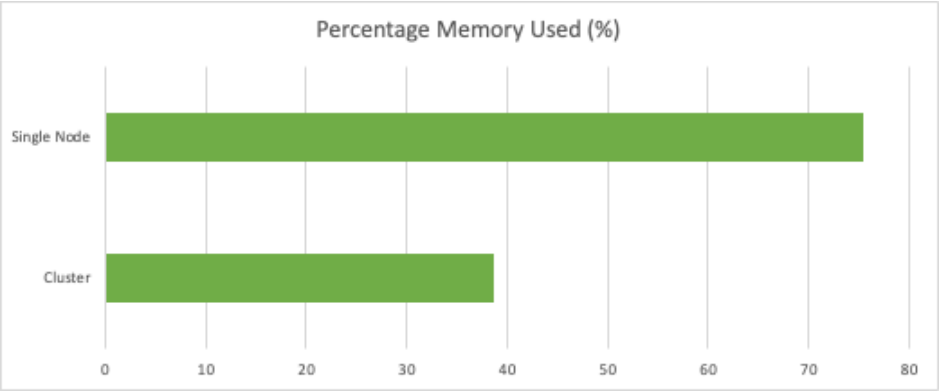

Figure 8.5: Percentage of memory used to total

8.6 Remarks

The most challenging issues in Federated Learning are strictly related to the Edge Computing devices and them nature.
In the proposed work, we can try to give a solution for some of them. The research proposes an innovative architecture in which the concept of Federated Learning Client is abstracted. With our solution, the client of Federated Learning architecture is decoupled and all the tasks involved are distributed in a smart way. This solution can bring more resource computational and can overcome the unbalanced distributed dataset issue that is frequent in Federated Learning and that can deteriorate the final accuracy of global model. The test performed in our work prove the positive trend in time response about the local model training.
In future works we plan to perform other tests about our solution, in particular, about the computational resource spent in our solution. Moreover, we wants to analyze the energy impact that our solution bring. In next works we want also to study the possibility to create a dynamic cluster FL architecture in which the clusters client are not established a priori but they can dynamically change in order to improve specific criteria of the model (accuracy, time, and so on).

CHAPTER 9

Conclusion and Future Works

The thesis deep Cloud Computing issues proposing several solutions to overcome the classical Cloud approach in order to solve the main Cloud Computing threats. It is divided into three parts:

- The first part treats the Security issues of Cloud Computing
- The second part cares about the issues related to the centralization of Cloud Computing
- The third part is focused on the issues that intersect the Security and the Distribution of the central approach.

For each part different solutions have been found, designed, realized and evaluated. In particular, for each point, two sides are considered: the user data and the computation. In Chapter 3 and 4 the first part is introduced and the Security of data management is deep. In particular, in this chapters, the security of data storage and secure elaboration has been the main topics. In Chapter 5 was deep and a solution for the distributed inference on Cloud-Edge environment is realized by exploiting different Workflows on Cloud-Edge Continuum. This solution introduce a complex operation like Artificial Intelligence Inference distributed among the Cloud and the Edge to overcome the problem related to the centralization of the processes typical in Cloud Infrastructure. In the end, the last three chapters investigate solutions for the issues that involve both Security and Distribution of Data and Computation. In particular in chapter 6 a solution for the Two-Factor Authentication is introduced. It allows to protection user access by distributing the logic of One-Time Password management

avoiding different security threats like the Distributed Deny of Service (DDoS). In chapter 7 another distributed solution is provided. In particular, here, a distributed Identity and Access Manager was designed and implemented. This kind of approach exploits Edge devices to perform a distributed and replicated authentication in network constraints environment, too. The last chapter of this final part was the chapter 8 in which a distributed solution for federated learning was introduced. In particular, in this chapter, an architecture for the clustering of each Federated Learning client was described, implemented and evaluated.

For future works, new solutions related to the total distribution of Machine Learning will be investigated. In particular, new solutions for decentralized Federated Learning and Distributed Training will be the object of the next studies in order to avoid at all, the use of a central infrastructure overcoming the computational limit of Edge tiny devices.

Bibliography

[1] Alessio Catalfamo, Armando Ruggeri, Antonio Celesti, Maria Fazio, and Massimo Villari. A microservices and blockchain based one time password (mbb-otp) protocol for security-enhanced authentication. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2021. (Cited at pages viii e 85)

[2] Alessio Catalfamo, Maria Fazio, Francesco Martella, Antonio Celesti, and Massimo Villari. Muovime: Secure access to sustainable mobility services in smart city. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–5, 2021. (Cited at page viii)

[3] Christian Sicari, Alessio Catalfamo, Antonino Galletta, and Massimo Villari. A distributed peer to peer identity and access management for the osmotic computing. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 775–781, 2022. (Cited at page viii)

[4] Alessio Catalfamo, Antonio Celesti, Maria Fazio, and Massimo Villari. A homomorphic encryption service to secure data processing in a cloud/edge continuum context. In *2022 9th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 55–61, 2022. (Cited at page viii)

[5] Alessio Catalfamo, Antonio Celesti, Maria Fazio, Giovanni Randazzo, and Massimo Villari. A platform for federated learning on the edge: a video analysis use case. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, 2022. (Cited at page ix)

[6] Alessio Catalfamo, Lorenzo Carnevale, Antonino Galletta, Francesco Martella, Antonio Celesti, Maria Fazio, and Massimo Villari. Scaling data analysis services in an edge-based federated learning environment. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 167–172, 2022. (Cited at page ix)

[7] Valeria Lukaj, Alessio Catalfamo, Francesco Martella, Maria Fazio, Massimo Villari, and Antonio Celesti. A nosql dbms transparent data encryption approach for cloud/edge continuum. (Cited at page ix)

[8] Alin Zamfiroiu, Ionut Petre, and Radu Boncea. Cloud computing vulnerabilities analysis. In *Proceedings of the 2019 4th International Conference on Cloud Computing and Internet of Things*, CCIOT '19, page 48–53, New York, NY, USA, 2019. Association for Computing Machinery. (Cited at page 1)

[9] Choi Myeonggil. The security risks of cloud computing. In *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 330–330, 2019. (Cited at page 1)

[10] Marwane Zekri, Said El Kafhali, Noureddine Aboutabit, and Youssef Saadi. Ddos attack detection using machine learning techniques in cloud computing environments. In *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*, pages 1–7, 2017. (Cited at page 1)

[11] Anil Acharya, Yantian Hou, Ying Mao, and Jiawei Yuan. Edge-assisted image processing with joint optimization of responding and placement strategy. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1241–1248, 2019. (Cited at page 2)

[12] Mohammad Manzurul Islam, Sarwar Morshed, and Parijat Goswami. Cloud computing: A survey on its limitations and potential solutions, 2013. (Cited at page 3)

[13] Yuan Cao and Lin Yang. A survey of identity management technology. In *2010 IEEE International Conference on Information Theory and Information Security*, pages 287–293, 2010. (Cited at page 6)

[14] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, and Kent

Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 5 2015. (Cited at page 7)

[15] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1592:223–238, 1999. (Cited at pages 15, 34 e 41)

[16] H Brendan McMahan Eider Moore Daniel Ramage Seth Hampson Blaise AgüeraAg and Agüera Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. 2017. (Cited at pages 17 e 108)

[17] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials*, 25(4):2983–3013, 2023. (Cited at page 17)

[18] Daniel Maniglia Amancio Da Silva and Rute C. Sofia. A discussion on context-awareness to better support the iot cloud/edge continuum. *IEEE Access*, 8:193686–193694, 2020. (Cited at page 21)

[19] Elahe Fazeldehkordi and Tor-Morten Grønli. A survey of security architectures for edge computing-based iot. *IoT*, 3(3):332–365, 2022. (Cited at page 22)

[20] Kewei Sha, T. Andrew Yang, Wei Wei, and Sadegh Davari. A survey of edge computing-based designs for iot security. *Digital Communications and Networks*, 6(2):195–202, 2020. (Cited at page 22)

[21] Valeria Lukaj, Francesco Martella, Maria Fazio, Antonio Celesti, and Massimo Villari. Trusted ecosystem for iot service provisioning based on brokering. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 746–753, 2021. (Cited at page 22)

[22] Lukaj Valeria, Martella Francesco, Fazio Maria, Celesti Antonio, and Villari Massimo. Establishment of a trusted environment for iot service provisioning based on x3dh-based brokering and federated blockchain. *Internet of Things*, 21:100686, 2023. (Cited at page 22)

[23] Qian Liu, Juan Gu, Jingchao Yang, Yun Li, Dexuan Sha, Mengchao Xu, Ishan Shams, Manzhu Yu, and Chaowei Yang. *Cloud, Edge, and Mobile Computing for Smart Cities*, pages 757–795. Springer Singapore, Singapore, 2021. (Cited at page 22)

[24] Evaristus Didik Madyatmadja, Aditya Nur Hakim, and David Jumpa Malem Sembiring. Performance testing on transparent data encryption for sql server's reliability and efficiency. *Journal of Big Data*, 8(1):134, 2021. (Cited at page 22)

[25] Dr. Anwar Pasha Deshmukh and Dr. Riyazuddin Qureshi. Transparent data encryption – solution for security of database contents. 2013. (Cited at page 22)

[26] K Natarajan and Vaheedbasha Shaik. Transparent data encryption: Comparative analysis and performance evaluation of oracle databases. In *2020 Fifth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pages 137–142, 2020. (Cited at page 22)

[27] Valeria Lukaj, Francesco Martella, Antonino Quattrocchi, Maria Fazio, Roberto Montanini, Massimo Villari, and Antonio Celesti. Towards iot rejuvenation: a study on hy-srf05 ultrasonic sensor ageing for intelligent street pole lamp control in a smart city. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–8, 2022. (Cited at page 22)

[28] N. Sai Sirisha and K. Venkat Kiran. Authorization of data in hadoop using apache sentry. *International Journal of Engineering & Technology*, 2018. (Cited at page 22)

[29] Devanshu Trivedi, Pavol Zavarsky, and Sergey Butakov. Enhancing relational database security by metadata segregation. *Procedia Computer Science*, 94:453–458, 2016. The 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops. (Cited at page 23)

[30] Vasily Sidorov and Wee Keong Ng. Transparent data encryption for data-in-use and data-at-rest in a cloud-based database-as-a-service solution. In *2015 IEEE World Congress on Services*, pages 221–228, 2015. (Cited at page 23)

[31] Joao Sousa, Tiago Mateus, and Pedro Furtado. Scalability architecture for a secure big data system. *International Journal of Business Process Integration and Management*, 7:345, 01 2015. (Cited at page 23)

[32] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001. (Cited at page 23)

[33] Na Su, Yi Zhang, and Mingyue Li. Research on data encryption standard based on aes algorithm in internet of things environment. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 2071–2075, 2019. (Cited at page 23)

[34] Pedro Sanchez Munoz, Nam Tran, Brandon Craig, Behnam Dezfouli, and Yuhong Liu. Analyzing the resource utilization of aes encryption on iot devices. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1200–1207, 2018. (Cited at page 23)

[35] Jiafu Wan, Jiapeng Li, Qingsong Hua, Antonio Celesti, and Zhongren Wang. Intelligent equipment design assisted by cognitive internet of things and industrial big data. *Neural Computing and Applications*, 32(9):4463–4472, 2020. (Cited at page 34)

[36] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. (Cited at pages 35 e 42)

[37] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. (Cited at pages 35 e 42)

[38] R L Rivest, A Shamir, and L Adleman. A method for obtaining digital signatures and public-key cryptosystems. (Cited at page 35)

[39] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. 1978. (Cited at page 35)

[40] A a survey on homomorphic encryption schemes: Theory and implementation. (Cited at page 35)

[41] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? (Cited at page 35)

[42] Guowen Xu, Yan Ren, Hongwei Li, Dongxiao Liu, Yuanshun Dai, and Kan Yang. Cryptmdb: A practical encrypted mongodb over big data. *IEEE International Conference on Communications*, 7 2017. (Cited at page 35)

[43] Baohua Huang, Sheng Liang, Dongdong Xu, and Zhuohao Wan. A homomorphic searching scheme for sensitive data in nosql database. *Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, iThings/Gree*, pages 575–579, 2018. (Cited at page 35)

[44] Xiaojie Guo, Jin Li, Zheli Liu, Yu Wei, Xiao Zhang, and Changyu Dong. Labrador: towards fair and auditable data sharing in cloud computing with long-term privacy. *Science China Information Sciences*, 65:152106, 5 2022. (Cited at page 36)

[45] Jean Louis Raisaro, Gwangbae Choi, Sylvain Pradervand, Raphael Colsenet, Nathalie Jacquemont, Nicolas Rosat, Vincent Mooser, and Jean Pierre Hubaux. Protecting privacy and security of genomic data in i2b2 with homomorphic encryption and differential privacy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15:1413–1426, 9 2018. (Cited at page 36)

[46] Koushik Sinha, Pratham Majumder, and Subhas K. Ghosh. Fully homomorphic encryption based privacy-preserving data acquisition and computation for contact tracing. *International Symposium on Advanced Networks and Telecommunication Systems, ANTS*, 2020-December, 12 2020. (Cited at page 36)

[47] Tanping Zhou, Wenchao Liu, Ningbo Li, Xiaoyuan Yang, Yiliang Han, and Shangwen Zheng. Secure scheme for locating disease-causing genes based on multi-key homomorphic encryption. *Tsinghua Science and Technology*, 27:333–343, 4 2022. (Cited at page 36)

[48] Ansar Rafique, Dimitri Van Landuyt, Emad Heydari Beni, Bert Lagaisse, and Wouter Joosen. Cryptdice: Distributed data protection system for secure cloud data storage and computation. *Information Systems*, 96:101671, 2 2021. Degree of trust che non possiamo dare per assodata nei servizi cloud moderni. (Cited at page 36)

[49] Python Paillier, `https://python-paillier.readthedocs.io/en/stable/index.html`, note = Last access Jan 2022. (Cited at page 41)

[50] PyFhel Library, `https://pyfhel.readthedocs.io/en/latest/`, note = Last access Jan 2022. (Cited at page 42)

[51] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. *Proceedings - International Conference on Distributed Computing Systems*, pages 328–339, 7 2017. (Cited at page 51)

[52] Arjun Parthasarathy and Bhaskar Krishnamachari. Defer: Distributed edge inference for deep neural networks. (Cited at page 51)

[53] Emad MalekHosseini, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahram Shirani. Splitting convolutional neural network structures for efficient inference, 2020. (Cited at page 51)

[54] Chenghao Hu and Baochun Li. Distributed inference with deep learning models across heterogeneous edge devices. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 330–339, 2022. (Cited at page 51)

[55] Rafael Stahl, Zhuoran Zhao, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. Fully distributed deep learning inference on resource-constrained edge devices. In Dionisios N. Pnevmatikatos, Maxime Pelcat, and Matthias Jung, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 77–90, Cham, 2019. Springer International Publishing. (Cited at page 51)

[56] Nicholas James, Lee-Yeng Ong, and Meng-Chew Leow. Exploring distributed deep learning inference using raspberry pi spark cluster. *Future Internet*, 14(8), 2022. (Cited at page 52)

[57] Anthony Thomas, Yunhui Guo, Yeseong Kim, Baris Aksanli, Arun Kumar, and Tajana S. Rosing. Hierarchical and distributed machine learning inference beyond the edge. In *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, pages 18–23, 2019. (Cited at page 52)

[58] C. Klingler, K. Schulz, and M. Herrnegger. Lamah-ce: Large-sample data for hydrology and environmental sciences for central europe. *Earth System Science Data*, 13(9):4529–4565, 2021. (Cited at page 53)

[59] G. Mehta, E. Deelman, K. Vahi, and F. Silva. Pegasus Workflow Management System: Helping Applications From Earth and Space. In *AGU Fall Meeting Abstracts*, volume 2010, pages IN41B–1362, December 2010. (Cited at page 57)

[60] Ulf Melin, Karin Axelsson, and Fredrik Söderström. Managing the development of e-id in a public e-service context. *Transforming Government: People, Process and Policy*, 2016. (Cited at page 66)

[61] Thanasis Petsas, Giorgos Tsirantonakis, Elias Athanasopoulos, and Sotiris Ioannidis. Two-factor authentication: is the world ready? quantifying 2fa adoption. In *Proceedings of the eighth european workshop on system security*, pages 1–7, 2015. (Cited at page 67)

[62] TOTP: Time-Based One-Time Password Algorithm, Available online: https://datatracker.ietf.org/doc/html/rfc6238 (accessed on 12 May 2021). (Cited at page 67)

[63] K Aravindhan and RR Karthiga. One time password: A survey. *International Journal of Emerging Trends in Engineering and Development*, 1(3):613–623, 2013. (Cited at page 68)

[64] Imamah. One Time Password (OTP) Based on Advanced Encrypted Standard (AES) and Linear Congruential Generator(LCG). *2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar, EECCIS 2018*, pages 391–394, 2018. (Cited at page 68)

[65] Badis Hammi, Achraf Fayad, Rida Khatoun, Sherali Zeadally, and Youcef Begriche. A Lightweight ECC-Based Authentication Scheme for Internet of Things (IoT). *IEEE Systems Journal*, 14(3):3440–3450, 2020. (Cited at page 69)

[66] A Strong Authentication Method for Web/Mobile Services. pages 124–129. Institute of Electrical and Electronics Engineers Inc., jun 2019. (Cited at page 69)

[67] Emir Erdem and Mehmet Tahir Sandikkaya. OTPaaS-One time password as a service. *IEEE Transactions on Information Forensics and Security*, 14(3):743–756, 2018. (Cited at page 69)

[68] Varun Amrutiya, Siddhant Jhamb, Pranjal Priyadarshi, and Ashutosh Bhatia. Trustless two-factor authentication using smart contracts in blockchains. *International Conference on Information Networking*, 2019-Janua(May):66–71, 2019. (Cited at page 69)

[69] Short Paper, Patrick Stewin, and Jean-pierre Seifert. SMS-Based One-Time Passwords : Attacks and Defense. pages 150–151, 2013. (Cited at page 69)

[70] Dimitrios Kallergis, Zacharenia Garofalaki, Georgios Katsikogiannis, and Christos Douligeris. Capodaz: A containerised authorisation and policy-driven architecture using microservices. *Ad Hoc Networks*, 104:102153, 2020. (Cited at page 69)

[71] Yustus Eko Oktian, Sang Gon Lee, and Hoon Jae Lee. TwoChain: Leveraging Blockchain and Smart Contract for Two Factor Authentication. In *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems, ISRITI 2020*, pages 187–191. Institute of Electrical and Electronics Engineers Inc., dec 2020. (Cited at page 69)

[72] Francesco Buccafurri, Vincenzo De Angelis, and Roberto Nardone. Securing MQTT by blockchain-based otp authentication. *Sensors (Switzerland)*, 20(7), 2020. (Cited at page 70)

[73] Eman Alharbi . and Daniyal Alghazzawi . Two Factor Authentication Framework Using OTP-SMS Based on Blockchain. *Transactions on Machine Learning and Artificial Intelligence*, 7(3), 2019. (Cited at page 70)

[74] Mingli Zhang, Liming Wang, and Jing Yang. A Blockchain-Based Authentication Method with One-Time Password. In *2019 IEEE 38th International Performance Computing and Communications Conference, IPCCC 2019*. Institute of Electrical and Electronics Engineers Inc., oct 2019. (Cited at page 70)

[75] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. Technical report, 2017. (Cited at pages 70 e 83)

[76] A. Celesti, M. Fazio, A. Galletta, L. Carnevale, J. Wan, and M. Villari. An approach for the secure management of hybrid cloud–edge environments. *Future Generation Computer Systems*, 90:1–19, 2019. (Cited at page 74)

[77] Miltiadis D Lytras and Anna Visvizi. Who uses smart city services and what to make of it: Toward interdisciplinary smart cities research. *Sustainability*, 10(6):1998, 2018. (Cited at page 82)

[78] Svein Ølnes, Jolien Ubacht, and Marijn Janssen. Blockchain in government: Benefits and implications of distributed ledger technology for information sharing. *Government Information Quarterly*, 34(3):355–364, 2017. (Cited at page 82)

[79] Armando Ruggeri, Maria Fazio, Antonio Celesti, and Massimo Villari. Blockchain-based healthcare workflows in federated hospital clouds. In *European Conference on Service-Oriented and Cloud Computing*, pages 113–121. Springer, 2020. (Cited at page 82)

[80] Claudia Antal, Tudor Cioara, Ionut Anghel, Marcel Antal, and Ioan Salomie. Distributed ledger technology review and decentralized applications development guidelines. *Future Internet*, 13(3):62, 2021. (Cited at page 82)

[81] Badr El Khalyly, Abdessamad Belangour, Mouad Banane, and Allae Erraissi. A comparative study of microservices-based iot platforms. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(7):389–398, 2020. (Cited at page 83)

[82] LXC, Available online: `https://linuxcontainers.org/` (accessed on 20 May 2021). (Cited at page 86)

[83] Infura, Available online: `https://infura.io/` (accessed on 20 May 2021). (Cited at page 88)

[84] C. Sicari, A. Galletta, A. Celesti, M. Fazio, and M. Villari. An osmotic computing enabled domain naming system (oce-dns) for distributed service relocation between cloud and edge. *Computers and Electrical Engineering*, 96, 2021. (Cited at page 92)

[85] Thomas Rausch, Schahram Dustdar, and Rajiv Ranjan. Osmotic message-oriented middleware for the internet of things. *IEEE Cloud Computing*, 5(2):17–25, 2018. (Cited at page 92)

[86] Anelis Pereira-Vale, Gastón Márquez, Hernán Astudillo, and Eduardo B. Fernandez. Security mechanisms used in microservices-based systems: A systematic mapping. *Proceedings - 2019 45th Latin American Computing Conference, CLEI 2019*, sep 2019. (Cited at page 93)

[87] Tetiana Yarygina and Anya Helene Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20, 2018. (Cited at page 93)

[88] A. Banati, E. Kail, K. Karoczkai, and M. Kozlovszky. Authentication and authorization orchestrator for microservice-based software architectures. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, pages 1180–1184, 6 2018. (Cited at page 93)

[89] Dimitrios Kallergis, Zacharenia Garofalaki, Georgios Katsikogiannis, and Christos Douligeris. Capodaz: A containerised authorisation and policy-driven architecture using microservices. *Ad Hoc Networks*, 104, 7 2020. (Cited at page 93)

[90] Daniel Richter, Tim Neumann, and Andreas Polze. Security considerations for microservice architectures. *CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science*, 2018-January:608–615, 2018. (Cited at page 94)

[91] Davy Preuveneers and Wouter Joosen. Access control with delegated authorization policy evaluation for data-driven microserviceworkflows. *Future Internet*, 9, 9 2017. (Cited at page 94)

[92] Anelis Pereira-Vale, Eduardo B. Fernandez, Raúl Monge, Hernán Astudillo, and Gastón Márquez. Security in microservice-based systems: A multivocal literature review. *Computers and Security*, 103, 4 2021. (Cited at page 94)

[93] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3:76–83, 11 2016. (Cited at pages 94 e 96)

[94] Alina Buzachis and Massimo Villari. Basic principles of osmotic computing: Secure and dependable microelements (mels) orchestration leveraging blockchain facilities. *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, pages 47–52, 1 2019. (Cited at page 94)

[95] Lorenzo Carnevale, Armando Ruggeri, Francesco Martella, Antonio Celesti, Maria Fazio, and Massimo Villari. Multi hop reconfiguration of end-devices in heterogeneous edge-iot mesh networks. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2021. (Cited at page 94)

[96] Pawel Szalachowski. Password-authenticated decentralized identities. *IEEE Transactions on Information Forensics and Security*, 16:4801–4810, 2021. (Cited at page 95)

[97] Meng Kang and Victoria Lemieux. A decentralized identity-based blockchain solution for privacy-preserving licensing of individual-controlled data to prevent unauthorized secondary data usage. *Ledger*, 6, 11 2021. (Cited at page 95)

[98] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel. A survey on essential components of a self-sovereign identity. *Computer Science Review*, 30:80–86, 11 2018. (Cited at page 95)

[99] Komal Gilani, Emmanuel Bertin, Julien Hatin, and Noel Crespi. A survey on blockchain-based identity management and decentralized privacy for personal data. *2020 2nd Conference on Blockchain Research and Applications for Innovative Networks and Services, BRAINS 2020*, pages 97–101, 9 2020. (Cited at page 95)

[100] Kubernetes DaemonSet, howpublished = `https://cloud.google.com/kubernetes-engine/docs/concepts/daemonset`, note = Last access May 2021. (Cited at pages 100 e 113)

[101] Antonino Galletta, Armando Ruggeri, Maria Fazio, Gianluca Dini, and Massimo Villari. Mesmart-pro: Advanced processing at the edge for smart urban monitoring and reconfigurable services. *Journal of Sensor and Actuator Networks*, 9(4), 2020. (Cited at page 102)

[102] Federated Learning - Google, url=`https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`, note = Last access Sep 2022. (Cited at page 108)

[103] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. *2015 53rd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2015*, pages 909–910, 4 2016. (Cited at page 108)

[104] Wei Liu, Li Chen, and Wenyi Zhang. Decentralized federated learning: Balancing communication and computing costs. *IEEE Transactions on Signal and Information Processing over Networks*, 8:131–143, 2022. (Cited at page 108)

[105] Ruchi Gupta and Tanweer Alam. Survey on Federated-Learning Approaches in Distributed Environment. *Wireless Personal Communications*, 2022. (Cited at page 108)

[106] H. Zhang, J. Bosch, and H.H. Olsson. Federated learning systems: Architecture alternatives. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, volume 2020-December, pages 385–394, 2020. (Cited at page 108)

[107] Collin Meese, Hang Chen, Syed Ali Asif, Wanxin Li, Chien-Chung Shen, and Mark Nejad. Bfrt: Blockchained federated learning for real-time traffic flow prediction. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 317–326, 2022. (Cited at page 108)

[108] Hassan Saadat, Abdulla Aboumadi, Amr Mohamed, Aiman Erbad, and Mohsen Guizani. Hierarchical federated learning for collaborative ids in iot applications. *2021*

10th Mediterranean Conference on Embedded Computing, MECO 2021, 6 2021. (Cited at page 108)

[109] Shen Xin, Li Zhuo, and Chen Xin. Online node cooperation strategy design for hierarchical federated learning. *INFOCOM WKSHPS 2022 - IEEE Conference on Computer Communications Workshops*, 2022. (Cited at page 108)

[110] Binxuan Hu, Yujia Gao, Liang Liu, and Huadong Ma. *Federated Region-Learning: An Edge Computing Based Framework for Urban Environment Sensing; Federated Region-Learning: An Edge Computing Based Framework for Urban Environment Sensing*. 2018. (Cited at page 109)

[111] Yujia Gao, Liang Liu, Binxuan Hu, Tianzi Lei, and Huadong Ma. Federated region-learning for environment sensing in edge computing system. *IEEE Transactions on Network Science and Engineering*, 7:2192–2204, 10 2020. (Cited at page 109)

[112] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. 5 2019. (Cited at page 109)

[113] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 11 2021. (Cited at page 109)

[114] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. (Cited at pages 109 e 113)

[115] Hyungbin Kim, Yongho Kim, and Hyunhee Park. Reducing model cost based on the weights of each layer for federated learning clustering. *International Conference on Ubiquitous and Future Networks, ICUFN*, 2021-August:405–408, 8 2021. (Cited at page 109)

[116] Cheng Chen, Ziyi Chen, Yi Zhou, and Bhavya Kailkhura. Fedcluster: Boosting the convergence of federated learning via cluster-cycling. *Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020*, pages 5017–5026, 12 2020. (Cited at page 109)

[117] Alessio Catalfamo, Antonio Celesti, Maria Fazio, Giovanni Randazzo, and Massimo Villari. A platform for federated learning on the edge: a video analysis use case. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, 2022. (Cited at page 109)

[118] Armando Ruggeri, Antonio Celesti, Maria Fazio, and Massimo Villari. An innovative blockchain-based orchestrator for osmotic computing. *Journal of Grid Computing*, 20(1):1–17, 2022. (Cited at page 109)

[119] A. Celesti, D. Mulfari, A. Galletta, M. Fazio, L. Carnevale, and M. Villari. A study on container virtualization for guarantee quality of service in Cloud-of-Things. *Future Generation Computer Systems*, 99, 2019. (Cited at page 110)

[120] GlusterFS network filesystem, howpublished = `https://www.gluster.org/`, note = Last access Oct 2022. (Cited at page 114)